<Network Working Group>
Internet Draft
Updates: 1964
Category: Standards Track
draft-ietf-krb-wg-gssapi-cfx-07.txt

Larry Zhu
Karthik Jaganathan
Microsoft
Sam Hartman
MIT

March 9, 2004
Expires: September 9, 2004

The Kerberos Version 5 GSS-API Mechanism: Version 2

Status of this Memo

Abstract

This document defines protocols, procedures, and conventions to be employed by peers implementing the Generic Security Service Application Program Interface (GSS-API) when using the Kerberos Version 5 mechanism.

RFC-1964 is updated and incremental changes are proposed in response to recent developments such as the introduction of Kerberos cryptosystem framework. These changes support the inclusion of new cryptosystems, by defining new per-message tokens along with their

encryption and checksum algorithms based on the cryptosystem profiles.

Conventions used in this document

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].

The term "little endian order" is used for brevity to refer to the least-significant-octet-first encoding, while the term "big endian order" is for the most-significant-octet-first encoding.

Table of Contents

1. Introduction

[KCRYPTO] defines a generic framework for describing encryption and checksum types to be used with the Kerberos protocol and associated protocols.

[RFC-1964] describes the GSS-API mechanism for Kerberos Version 5. It defines the format of context establishment, per-message and context deletion tokens and uses algorithm identifiers for each cryptosystem in per message and context deletion tokens.

The approach taken in this document obviates the need for algorithm identifiers. This is accomplished by using the same encryption algorithm, specified by the crypto profile [KCRYPTO] for the session key or subkey that is created during context negotiation, and its required checksum algorithm. Message layouts of the per-message

tokens are therefore revised to remove algorithm indicators and also to add extra information to support the generic crypto framework [KCRYPTO].

Tokens transferred between GSS-API peers for security context establishment are also described in this document. The data elements exchanged between a GSS-API endpoint implementation and the Kerberos Key Distribution Center (KDC) [KRBCLAR] are not specific to GSS-API usage and are therefore defined within [KRBCLAR] rather than within this specification.

The new token formats specified in this document MUST be used with all "newer" encryption types [KRBCLAR] and MAY be used with "older" encryption types, provided that the initiator and acceptor know, from the context establishment, that they can both process these new token formats.

"Newer" encryption types are those which have been specified along with or since the new Kerberos cryptosystem specification [KCRYPTO], as defined in section 3.1.3 of [KRBCLAR]. The list of not-newer encryption types is as follows [KCRYPTO]:

Encryption Type	Assigned Number
des-cbc-crc	1
des-cbc-md4	2
des-cbc-md5	3
des3-cbc-md5	5
des3-cbc-sha1	7
dsaWithSHA1-CmsOID	9
md5WithRSAEncryption-CmsOID	10
sha1WithRSAEncryption-CmsOID	11
rc2CBC-EnvOID	12
rsaEncryption-EnvOID	13

rsaES-OAEP-ENV-OID	14
des-ede3-cbc-Env-OID	15
des3-cbc-sha1-kd	16
rc4-hmac	23

2. Key Derivation for Per-Message Tokens

To limit the exposure of a given key, [KCRYPTO] adopted "one-way" "entropy-preserving" derived keys, for different purposes or key usages, from a base key or protocol key.

This document defines four key usage values below that are used to derive a specific key for signing and sealing messages, from the session key or subkey [KRBCLAR] created during the context establishment.

Name	Value
KG-USAGE-ACCEPTOR-SEAL	22
KG-USAGE-ACCEPTOR-SIGN	23
KG-USAGE-INITIATOR-SEAL	24

KG-USAGE-INITIATOR-SIGN	25

When the sender is the context acceptor, KG-USAGE-ACCEPTOR-SIGN is used as the usage number in the key derivation function for deriving keys to be used in MIC tokens (as defined in section 4.2.6.1), and KG-USAGE-ACCEPTOR-SEAL is used for Wrap tokens(as defined in section 4.2.6.2); similarly when the sender is the context initiator, KG-USAGE-INITIATOR-SIGN is used as the usage number in the key derivation function for MIC tokens, KG-USAGE-INITIATOR-SEAL is used for Wrap Tokens. Even if the Wrap token does not provide for confidentiality the same usage values specified above are used.

During the context initiation and acceptance sequence, the acceptor MAY assert a subkey, and if so, subsequent messages MUST use this subkey as the protocol key and these messages MUST be flagged as "AcceptorSubkey" as described in section 4.2.2.

3. Quality of Protection

The GSS-API specification [RFC-2743] provides for Quality of Protection (QOP) values that can be used by applications to request a certain type of encryption or signing. A zero QOP value is used to indicate the "default" protection; applications which do not use the default QOP are not guaranteed to be portable across implementations or even inter-operate with different deployment configurations of the same implementation. Using an algorithm that

is different from the one for which the key is defined may not be appropriate. Therefore, when the new method in this document is used, the QOP value is ignored.

The encryption and checksum algorithms in per-message tokens are now implicitly defined by the algorithms associated with the session key or subkey. Algorithms identifiers as described in [RFC-1964] are therefore no longer needed and removed from the new token headers.

4. Definitions and Token Formats

This section provides terms and definitions, as well as descriptions for tokens specific to the Kerberos Version 5 GSS-API mechanism.

4.1. Context Establishment Tokens

All context establishment tokens emitted by the Kerberos Version 5 GSS-API mechanism SHALL have the framing described in section 3.1 of [RFC-2743], as illustrated by the following pseudo-ASN.1 structures:

```
        GSS-API DEFINITIONS ::=

        BEGIN

        MechType ::= OBJECT IDENTIFIER
        -- representing Kerberos V5 mechanism

        GSSAPI-Token ::=
        -- option indication (delegation, etc.) indicated within
```



```
        -- mechanism-specific token
        [APPLICATION 0] IMPLICIT SEQUENCE {
                thisMech MechType,
                innerToken ANY DEFINED BY thisMech
                   -- contents mechanism-specific
                   -- ASN.1 structure not required
                }

        END
```

Where the innerToken field starts with a two-octet token-identifier (TOK_ID) expressed in big endian order, followed by a Kerberos message.

Here are the TOK_ID values used in the context establishment tokens:

Token	TOK_ID Value in Hex
KRB_AP_REQ	01 00

KRB_AP_REP	02 00
KRB_ERROR	03 00

Where Kerberos message KRB_AP_REQUEST, KRB_AP_REPLY, and KRB_ERROR are defined in [KRBCLAR].

If an unknown token identifier (TOK_ID) is received in the initial context establishment token, the receiver MUST return GSS_S_CONTINUE_NEEDED major status, and the returned output token MUST contain a KRB_ERROR message with the error code KRB_AP_ERR_MSG_TYPE [KRBCLAR].

4.1.1. Authenticator Checksum

The authenticator in the KRB_AP_REQ message MUST include the optional sequence number and the checksum field. The checksum field is used to convey service flags, channel bindings, and optional delegation information.

The checksum type MUST be 0x8003. When delegation is used, a ticket-granting ticket will be transferred in a KRB_CRED message. This ticket SHOULD have its forwardable flag set. The EncryptedData field of the KRB_CRED message [KRBCLAR] MUST be encrypted in the session key of the ticket used to authenticate the context.

The authenticator checksum field SHALL have the following format:

Octet	Name	Description
0..3	Lgth	Number of octets in Bnd field; Represented in little-endian order; Currently contains hex value 10 00 00 00 (16).
4..19	Bnd	Channel binding information, as described in section 4.1.1.2.
20..23	Flags	Four-octet context-establishment flags in little-endian order as described in section 4.1.1.1.
24..25	DlgOpt	The delegation option identifier (=1) in little-endian order [optional]. This field and the next two fields are present if and only if GSS_C_DELEG_FLAG is set as described in section 4.1.1.1.
26..27	Dlgth	The length of the Deleg field in little-endian order [optional].
28..(n-1)	Deleg	A KRB_CRED message (n = Dlgth + 28) [optional].
n..last	Exts	Extensions [optional].

The length of the checksum field MUST be at least 24 octets when GSS_C_DELEG_FLAG is not set (as described in section 4.1.1.1), and at least 28 octets plus Dlgth octets when GSS_C_DELEG_FLAG is set. When GSS_C_DELEG_FLAG is set, the DlgOpt, Dlgth and Deleg fields of the checksum data MUST immediately follow the Flags field. The optional trailing octets (namely the "Exts" field) facilitate future extensions to this mechanism. When delegation is not used but the Exts field is present, the Exts field starts at octet 24 (DlgOpt, Dlgth and Deleg are absent).

Initiators that do not support the extensions MUST NOT include more than 24 octets in the checksum field, when GSS_C_DELEG_FLAG is not set, or more than 28 octets plus the KRB_CRED in the Deleg field, when GSS_C_DELEG_FLAG is set. Acceptors that do not understand the extensions MUST ignore any octets past the Deleg field of the checksum data, when GSS_C_DELEG_FLAG is set, or past the Flags field of the checksum data, when GSS_C_DELEG_FLAG is not set.

4.1.1.1. Checksum Flags Field

The checksum "Flags" field is used to convey service options or extension negotiation information.

The following context establishment flags are defined in [RFC-2744].

Flag Name	Value
GSS_C_DELEG_FLAG	1
GSS_C_MUTUAL_FLAG	2
GSS_C_REPLAY_FLAG	4
GSS_C_SEQUENCE_FLAG	8
GSS_C_CONF_FLAG	16
GSS_C_INTEG_FLAG	32

Context establishment flags are exposed to the calling application. If the calling application desires a particular service option then it requests that option via GSS_Init_sec_context() [RFC-2743]. If the corresponding return state values [RFC-2743] indicate that any of above optional context level services will be active on the context, the corresponding flag values in the table above MUST be set in the checksum Flags field.

Flag values 4096..524288 (2^12, 2^13, ..., 2^19) are reserved for use with legacy vendor-specific extensions to this mechanism.

All other flag values not specified herein are reserved for future use. Future revisions of this mechanism may use these reserved

flags and may rely on implementations of this version to not use such flags in order to properly negotiate mechanism versions. Undefined flag values MUST be cleared by the sender, and unknown flags MUST be ignored by the receiver.

4.1.1.2. Channel Binding Information

These tags are intended to be used to identify the particular communications channel for which the GSS-API security context establishment tokens are intended, thus limiting the scope within which an intercepted context establishment token can be reused by an attacker (see [RFC-2743], section 1.1.6).

When using C language bindings, channel bindings are communicated to the GSS-API using the following structure [RFC-2744]:

```
typedef struct gss_channel_bindings_struct {
   OM_uint32       initiator_addrtype;
   gss_buffer_desc initiator_address;
   OM_uint32       acceptor_addrtype;
   gss_buffer_desc acceptor_address;
   gss_buffer_desc application_data;
} *gss_channel_bindings_t;
```

The member fields and constants used for different address types are defined in [RFC-2744].

The "Bnd" field contains the MD5 hash of channel bindings, taken over all non-null components of bindings, in order of declaration. Integer fields within channel bindings are represented in little-endian order for the purposes of the MD5 calculation.

In computing the contents of the Bnd field, the following detailed points apply:

(1) For purposes of MD5 hash computation, each integer field and input length field SHALL be formatted into four octets, using little endian octet ordering.

(2) All input length fields within gss_buffer_desc elements of a gss_channel_bindings_struct even those which are zero-valued, SHALL be included in the hash calculation; the value elements of gss_buffer_desc elements SHALL be dereferenced, and the resulting data SHALL be included within the hash computation, only for the case of gss_buffer_desc elements having non-zero length specifiers.

(3) If the caller passes the value GSS_C_NO_BINDINGS instead of a valid channel binding structure, the Bnd field SHALL be set to 16 zero-valued octets.

If the caller to GSS_Accept_sec_context [RFC-2743] passes in GSS_C_NO_CHANNEL_BINDINGS [RFC-2744] as the channel bindings then the acceptor MAY ignore any channel bindings supplied by the initiator, returning success even if the initiator did pass in channel bindings.

If the application supply, in the channel bindings, a buffer with a length field larger than 4294967295 (2^32 - 1), the implementation of this mechanism MAY chose to reject the channel bindings altogether, using major status GSS_S_BAD_BINDINGS [RFC-2743]. In any case, the size of channel binding data buffers that can be used (interoperable, without extensions) with this specification is limited to 4294967295 octets.

4.2. Per-Message Tokens

Two classes of tokens are defined in this section: "MIC" tokens, emitted by calls to GSS_GetMIC() and consumed by calls to GSS_VerifyMIC(), "Wrap" tokens, emitted by calls to GSS_Wrap() and consumed by calls to GSS_Unwrap().

The new per-message tokens introduced here do not include the generic GSS-API token framing used by the context establishment tokens. These new tokens are designed to be used with newer crypto systems that can, for example, have variable-size checksums.

4.2.1. Sequence Number

To distinguish intentionally-repeated messages from maliciously-replayed ones, per-message tokens contain a sequence number field, which is a 64 bit integer expressed in big endian order. After sending a GSS_GetMIC() or GSS_Wrap() token, the sender's sequence numbers SHALL be incremented by one.

4.2.2. Flags Field

The "Flags" field is a one-octet integer used to indicate a set of attributes for the protected message. For example, one flag is allocated as the direction-indicator, thus preventing an adversary from sending back the same message in the reverse direction and having it accepted.

The meanings of bits in this field (the least significant bit is bit 0) are as follows:

Bit	Name	Description
0	SentByAcceptor	When set, this flag indicates the sender is the context acceptor. When not set,

		it indicates the sender is the context initiator.
1	Sealed	When set in Wrap tokens, this flag indicates confidentiality is provided for. It SHALL NOT be set in MIC tokens.
2	AcceptorSubkey	A subkey asserted by the context acceptor

is used to protect the message.

The rest of available bits are reserved for future use and MUST be cleared. The receiver MUST ignore unknown flags.

4.2.3. EC Field

The "EC" (Extra Count) field is a two-octet integer field expressed in big endian order.

In Wrap tokens with confidentiality, the EC field SHALL be used to encode the number of octets in the filler, as described in section 4.2.4.

In Wrap tokens without confidentiality, the EC field SHALL be used to encode the number of octets in the trailing checksum, as described in section 4.2.4.

4.2.4. Encryption and Checksum Operations

The encryption algorithms defined by the crypto profiles provide for integrity protection [KCRYPTO]. Therefore no separate checksum is needed.

The result of decryption can be longer than the original plaintext [KCRYPTO] and the extra trailing octets are called "crypto-system residue" in this document. However, given the size of any plaintext data, one can always find a (possibly larger) size so that, when padding the to-be-encrypted text to that size, there will be no crypto-system residue added [KCRYPTO].

In Wrap tokens that provide for confidentiality, the first 16 octets of the Wrap token (the "header", as defined in section 4.2.6), SHALL be appended to the plaintext data before encryption. Filler octets MAY be inserted between the plaintext data and the "header", and the values and size of the filler octets are chosen by implementations, such that there SHALL be no crypto-system residue present after the decryption. The resulting Wrap token is {"header" | encrypt(plaintext-data | filler | "header")}, where encrypt() is the encryption operation (which provides for integrity protection) defined in the crypto profile [KCRYPTO], and the RRC field (as defined in section 4.2.5) in the to-be-encrypted header contain the

hex value 00 00.

In Wrap tokens that do not provide for confidentiality, the checksum SHALL be calculated first over the to-be-signed plaintext data, and then the first 16 octets of the Wrap token (the "header", as defined in section 4.2.6). Both the EC field and the RRC field in the token header SHALL be filled with zeroes for the purpose of calculating the checksum. The resulting Wrap token is {"header" | plaintext-data | get_mic(plaintext-data | "header")}, where get_mic() is the checksum operation for the required checksum mechanism of the chosen encryption mechanism defined in the crypto profile [KCRYPTO].

The parameters for the key and the cipher-state in the encrypt() and get_mic() operations have been omitted for brevity.

For MIC tokens, the checksum SHALL be calculated as follows: the checksum operation is calculated first over the to-be-signed plaintext data, and then the first 16 octets of the MIC token, where the checksum mechanism is the required checksum mechanism of the chosen encryption mechanism defined in the crypto profile [KCRYPTO].

The resulting Wrap and MIC tokens bind the data to the token header, including the sequence number and the direction indicator.

4.2.5. RRC Field

The "RRC" (Right Rotation Count) field in Wrap tokens is added to allow the data to be encrypted in-place by existing SSPI (Security Service Provider Interface) [SSPI] applications that do not provide an additional buffer for the trailer (the cipher text after the in-place-encrypted data) in addition to the buffer for the header (the cipher text before the in-place-encrypted data). The resulting Wrap token in the previous section, excluding the first 16 octets of the token header, is rotated to the right by "RRC" octets. The net result is that "RRC" octets of trailing octets are moved toward the header. Consider the following as an example of this rotation operation: Assume that the RRC value is 3 and the token before the rotation is {"header" | aa | bb | cc | dd | ee | ff | gg | hh}, the token after rotation would be {"header" | ff | gg | hh | aa | bb | cc | dd | ee }, where {aa | bb | cc |...| hh} is used to indicate the octet sequence.

The RRC field is expressed as a two-octet integer in big endian order.

The rotation count value is chosen by the sender based on implementation details, and the receiver MUST be able to interpret

all possible rotation count values, including rotation counts greater than the length of the token.

4.2.6. Message Layouts

Per-message tokens start with a two-octet token identifier (TOK_ID) field, expressed in big endian order. These tokens are defined separately in subsequent sub-sections.

4.2.6.1. MIC Tokens

Use of the GSS_GetMIC() call yields a token (referred as the MIC token in this document), separate from the user data being protected, which can be used to verify the integrity of that data as received. The token has the following format:

Octet no	Name	Description
0..1	TOK_ID	Identification field. Tokens emitted by GSS_GetMIC() contain the hex value 04 04 expressed in big endian order in this field.
2	Flags	Attributes field, as described in section 4.2.2.
3..7	Filler	Contains five octets of hex value FF.
8..15	SND_SEQ	Sequence number field in clear text, expressed in big endian order.
16..last	SGN_CKSUM	Checksum of the "to-be-signed" data and octet 0..15, as described in section 4.2.4.

The Filler field is included in the checksum calculation for simplicity.

4.2.6.2. Wrap Tokens

Use of the GSS_Wrap() call yields a token (referred as the Wrap token in this document), which consists of a descriptive header, followed by a body portion that contains either the input user data in plaintext concatenated with the checksum, or the input user data encrypted. The GSS_Wrap() token SHALL have the following format:

Octet no	Name	Description
0..1	TOK_ID	Identification field. Tokens emitted by GSS_Wrap() contain the the hex value 05 04 expressed in big endian order in this field.
2	Flags	Attributes field, as described in section 4.2.2.
3	Filler	Contains the hex value FF.

4..5	EC	Contains the "extra count" field, in big endian order as described in section 4.2.3.
6..7	RRC	Contains the "right rotation count" in big endian order, as described in section 4.2.5.
8..15	SND_SEQ	Sequence number field in clear text, expressed in big endian order.
16..last	Data	Encrypted data for Wrap tokens with confidentiality, or plaintext data followed by the checksum for Wrap tokens without confidentiality, as described in section 4.2.4.

4.3. Context Deletion Tokens

Context deletion tokens are empty in this mechanism. Both peers to a security context invoke GSS_Delete_sec_context() [RFC-2743] independently, passing a null output_context_token buffer to indicate that no context_token is required. Implementations of GSS_Delete_sec_context() should delete relevant locally-stored context information.

4.4. Token Identifier Assignment Considerations

Token identifiers (TOK_ID) from 0x60 0x00 through 0x60 0xFF inclusive are reserved and SHALL NOT be assigned. Thus by examining the first two octets of a token, one can tell unambiguously if it is wrapped with the generic GSS-API token framing.

5. Parameter Definitions

This section defines parameter values used by the Kerberos V5 GSS-API mechanism. It defines interface elements in support of portability, and assumes use of C language bindings per [RFC-2744].

5.1. Minor Status Codes

This section recommends common symbolic names for minor_status values to be returned by the Kerberos V5 GSS-API mechanism. Use of these definitions will enable independent implementers to enhance application portability across different implementations of the mechanism defined in this specification. (In all cases, implementations of GSS_Display_status() will enable callers to convert minor_status indicators to text representations.) Each implementation should make available, through include files or other means, a facility to translate these symbolic names into the concrete values which a particular GSS-API implementation uses to represent the minor_status values specified in this section.

It is recognized that this list may grow over time, and that the need for additional minor_status codes specific to particular implementations may arise. It is recommended, however, that implementations should return a minor_status value as defined on a mechanism-wide basis within this section when that code is accurately representative of reportable status rather than using a separate, implementation-defined code.

5.1.1. Non-Kerberos-specific codes

```
GSS_KRB5_S_G_BAD_SERVICE_NAME
        /* "No @ in SERVICE-NAME name string" */
GSS_KRB5_S_G_BAD_STRING_UID
        /* "STRING-UID-NAME contains nondigits" */
GSS_KRB5_S_G_NOUSER
        /* "UID does not resolve to username" */
GSS_KRB5_S_G_VALIDATE_FAILED
        /* "Validation error" */
GSS_KRB5_S_G_BUFFER_ALLOC
        /* "Couldn't allocate gss_buffer_t data" */
GSS_KRB5_S_G_BAD_MSG_CTX
        /* "Message context invalid" */
GSS_KRB5_S_G_WRONG_SIZE
        /* "Buffer is the wrong size" */
GSS_KRB5_S_G_BAD_USAGE
        /* "Credential usage type is unknown" */
GSS_KRB5_S_G_UNKNOWN_QOP
        /* "Unknown quality of protection specified" */
```

5.1.2. Kerberos-specific-codes

```
GSS_KRB5_S_KG_CCACHE_NOMATCH
        /* "Client principal in credentials does not match
           specified name" */
```



```
GSS_KRB5_S_KG_KEYTAB_NOMATCH
        /* "No key available for specified service principal" */
GSS_KRB5_S_KG_TGT_MISSING
        /* "No Kerberos ticket-granting ticket available" */
GSS_KRB5_S_KG_NO_SUBKEY
        /* "Authenticator has no subkey" */
GSS_KRB5_S_KG_CONTEXT_ESTABLISHED
        /* "Context is already fully established" */
GSS_KRB5_S_KG_BAD_SIGN_TYPE
        /* "Unknown signature type in token" */
GSS_KRB5_S_KG_BAD_LENGTH
        /* "Invalid field length in token" */
GSS_KRB5_S_KG_CTX_INCOMPLETE
        /* "Attempt to use incomplete security context" */
```

5.2. Buffer Sizes

All implementations of this specification MUST be capable of accepting buffers of at least 16K octets as input to GSS_GetMIC(), GSS_VerifyMIC(), and GSS_Wrap(), and MUST be capable of accepting the output_token generated by GSS_Wrap() for a 16K octet input buffer as input to GSS_Unwrap(). Implementations SHOULD support 64K octet input buffers, and MAY support even larger input buffer sizes.

6. Backwards Compatibility Considerations

The new token formats defined in this document will only be recognized by new implementations. To address this, implementations can always use the explicit sign or seal algorithm in [RFC-1964] when the key type corresponds to "older" enctypes. An alternative approach might be to retry sending the message with the sign or seal algorithm explicitly defined as in [RFC-1964]. However this would require either the use of a mechanism such as [RFC-2478] to securely negotiate the method or the use out of band mechanism to choose appropriate mechanism. For this reason, it is RECOMMENDED that the new token formats defined in this document SHOULD be used only if both peers are known to support the new mechanism during context negotiation because of, for example, the use of "new" enctypes.

GSS_Unwrap() or GSS_VerifyMIC() can process a message token as follows: it can look at the first octet of the token header, if it is 0x60 then the token must carry the generic GSS-API pseudo ASN.1 framing, otherwise the first two octets of the token contain the TOK_ID that uniquely identify the token message format.

7. Security Considerations

Channel bindings are validated by the acceptor. The acceptor can ignore the channel bindings restriction supplied by the initiator and carried in the authenticator checksum, if channel bindings are not used by GSS_Accept_sec_context [RFC-2743], and the acceptor does not prove to the initiator that it has the same channel bindings as the initiator, even if the client requested mutual authentication. This limitation should be taken into consideration by designers of applications that would use channel bindings, whether to limit the

use of GSS-API contexts to nodes with specific network addresses, to authenticate other established, secure channels using Kerberos Version 5, or for any other purpose.

Session key types are selected by the KDC. Under the current mechanism, no negotiation of algorithm types occurs, so server-side (acceptor) implementations cannot request that clients not use

algorithm types not understood by the server. However, administrators can control what enctypes can be used for session keys for this mechanism by controlling the set of the ticket session key enctypes which the KDC is willing to use in tickets for a given acceptor principal. The KDC could therefore be given the task of limiting session keys for a given service to types actually supported by the Kerberos and GSSAPI software on the server. This does have a drawback for cases where a service principal name is used both for GSSAPI-based and non-GSSAPI-based communication (most notably the "host" service key), if the GSSAPI implementation does not understand (for example) AES [AES-KRB5] but the Kerberos implementation does. It means that AES session keys cannot be issued for that service principal, which keeps the protection of non-GSSAPI services weaker than necessary. KDC administrators desiring to limit the session key types to support interoperability with such GSSAPI implementations should carefully weigh the reduction in protection offered by such mechanisms against the benefits of interoperability.

8. Acknowledgments

Ken Raeburn and Nicolas Williams corrected many of our errors in the use of generic profiles and were instrumental in the creation of this document.

The text for security considerations was contributed by Nicolas Williams and Ken Raeburn.

Sam Hartman and Ken Raeburn suggested the "floating trailer" idea, namely the encoding of the RRC field.

Sam Hartman and Nicolas Williams recommended the replacing our earlier key derivation function for directional keys with different key usage numbers for each direction as well as retaining the directional bit for maximum compatibility.

Paul Leach provided numerous suggestions and comments.

Scott Field, Richard Ward, Dan Simon, Kevin Damour, and Simon Josefsson also provided valuable inputs on this document.

Jeffrey Hutzelman provided comments and clarifications for the text related to the channel bindings.

Jeffrey Hutzelman and Russ Housley suggested many editorial changes.

Luke Howard provided implementations of this document for the Heimdal code base, and helped inter-operability testing with the Microsoft code base, together with Love Hornquist Astrand. These experiments formed the basis of this document.

Martin Rex provided suggestions of TOK_ID assignment recommendations thus the token tagging in this document is unambiguous if the token is wrapped with the pseudo ASN.1 header.

John Linn wrote the original Kerberos Version 5 mechanism specification [RFC-1964], of which some of the text has been retained in this document.

9. Intellectual Property Statement

10. References

10.1. Normative References

[RFC-2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[RFC-2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC-2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC-2744] Wray, J., "Generic Security Service API Version 2: C-bindings", RFC 2744, January 2000.

[RFC-1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[KCRYPTO] RFC-Editor: To be replaced by RFC number for draft-ietf-krb-wg-crypto. Work in Progress.

[KRBCLAR] RFC-Editor: To be replaced by RFC number for draft-ietf-krb-wg-kerberos-clarifications. Work in Progress.

10.2. Informative References

[SSPI] Leach, P., "Security Service Provider Interface", Microsoft Developer Network (MSDN), April 2003.

[AES-KRB5] RFC-Editor: To be replaced by RFC number for draft-raeburn-krb-rijndael-krb. Work in Progress.

[RFC-2478] Baize, E., Pinkas D., "The Simple and Protected GSS-API Negotiation Mechanism", RFC 2478, December 1998.

11. Author's Address

Larry Zhu
One Microsoft Way
Redmond, WA 98052 - USA
EMail: LZhu@microsoft.com

Karthik Jaganathan
One Microsoft Way
Redmond, WA 98052 - USA
EMail: karthikj@microsoft.com

Sam Hartman
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139 - USA
Email: hartmans@MIT.EDU

Full Copyright Statement