1. Abstract

[RFC-1964] defines protocols, procedures, and conventions to be employed by peers implementing the Generic Security Service Application Program Interface (as specified in [RFC-2743]) when using the Kerberos Version 5 mechanism (as specified in [KRBCLEAR]).

This memo obsoletes [RFC-1964] and proposes changes in response to recent developments such as the introduction of Kerberos crypto framework. It is intended that this memo or a subsequent version will become the Kerberos Version 5 GSS-API mechanism specification on the standards track.

2. 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].

3. 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 GSSAPI mechanism for Kerberos V5. It defines the format of context initiation, 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 and checksum algorithms specified by the crypto profile [KCRYPTO] for the session key or subkey that is created during context negotiation. Message layouts of the per-message and context deletion 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 initiation are also described in this document. The data elements exchanged between a GSS-API endpoint implementation and the Kerberos KDC are not specific to GSS-API usage and are therefore defined within [KRBCLEAR] rather than within this specification.

The new token formats specified in this memo MUST be used with all "newer" encryption types [KRBCLEAR] 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] [KRBCLEAR].

Note that in this document, "AES" is used for brevity to refer loosely to either aes128-cts-hmac-sha1-96 or aes256-cts-hmac-sha1-96 as defined in [AES-KRB5]. AES is used as an example of the new method defined in this document.

4. Key Derivation for Per-Message and Context Deletion 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 for signing and sealing messages:

<table>
<thead>
<tr>
<th>Name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KG-USAGE-ACCEPTOR-SEAL</td>
<td>22</td>
</tr>
<tr>
<td>KG-USAGE-ACCEPTOR-SIGN</td>
<td>23</td>
</tr>
<tr>
<td>KG-USAGE-INITIATOR-SEAL</td>
<td>24</td>
</tr>
<tr>
<td>KG-USAGE-INITIATOR-SIGN</td>
<td>25</td>
</tr>
</tbody>
</table>

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 and context deletion tokens, and KG-USAGE-ACCEPTOR-SEAL is used for Wrap tokens; 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 and context deletion 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.
5. Quality of Protection

The GSSAPI specification [RFC-2743] provides for Quality of Protection (QOP) values that can be used by the application to request a certain type of encryption or signing. A zero QOP value is used to indicate the "default" protection; applications which 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 and context deletion 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.

6. Token Framing

Per [RFC-2743], all tokens emitted by the Kerberos V5 GSS-API mechanism will have the framing shown below:

```
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
```

The innerToken field always starts with a two byte token-identifier (TOK_ID). Here are the TOK_ID values:

<table>
<thead>
<tr>
<th>Token</th>
<th>TOK_ID Value in hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRB_AP_REQUEST</td>
<td>01 00</td>
</tr>
<tr>
<td>KRB_AP_REQPLY</td>
<td>02 00</td>
</tr>
<tr>
<td>KRB_ERROR</td>
<td>03 00</td>
</tr>
<tr>
<td>MIC</td>
<td>01 01</td>
</tr>
<tr>
<td>Wrap</td>
<td>01 02</td>
</tr>
<tr>
<td>[RFC-1964] context deletion</td>
<td>02 01</td>
</tr>
<tr>
<td>MIC</td>
<td>04 04</td>
</tr>
<tr>
<td>Wrap</td>
<td>04 05</td>
</tr>
</tbody>
</table>
7. Context Initialization Tokens

For context initialization tokens, the body for the innerToken field contains a Kerberos V5 message (KRB_AP_REQUEST, KRB_AP_REPLY, or KRB_ERROR) as defined in [KRBCLR].

7.1. 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 binding, and optional delegation information. It MUST have a type of 0x8003. The length of the checksum MUST be 24 bytes when delegation is not used. When delegation is used, a TGT with its FORWARDABLE flag set will be transferred within the KRB_CRED [KRBCLR] message.

The format of the authenticator checksum field is as follows.

<table>
<thead>
<tr>
<th>Byte</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..3</td>
<td>Lgth</td>
<td>Number of bytes in Bnd field; Currently contains hex 10 00 00 00 (16, represented in little-endian form)</td>
</tr>
<tr>
<td>4..19</td>
<td>Bnd</td>
<td>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.</td>
</tr>
<tr>
<td>20..23</td>
<td>Flags</td>
<td>Bit vector of context-establishment flags, as defined next. The resulting bit vector is encoded into bytes 20..23 in little-endian form.</td>
</tr>
<tr>
<td>24..25</td>
<td>DlgOpt</td>
<td>The Delegation Option identifier (=1) [optional]</td>
</tr>
<tr>
<td>26..27</td>
<td>Dlgth</td>
<td>The length of the Deleg field [optional]</td>
</tr>
<tr>
<td>28..n</td>
<td>Deleg</td>
<td>A KRB_CRED message (n = Dlgth + 29) [optional]</td>
</tr>
</tbody>
</table>

[we need to get input on how to allow additional data for extensions. Nicolas will post some text for this. If that is the case, do we need to change the checksum type?]

7.1.1. Flags Field

The checksum flags are used to convey service options or extension negotiation information. The bits in the Flags field are allocated as follows (Most significant bit is bit 0):

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..11</td>
<td>Mandatory</td>
<td>Critical extension flags</td>
</tr>
<tr>
<td>12..15</td>
<td>Optional</td>
<td>Non-critical extension flags</td>
</tr>
<tr>
<td>16..31</td>
<td>Standard</td>
<td>Context establishment flags</td>
</tr>
</tbody>
</table>

An extension or context establishment flag can either be critical or non-critical. When the context initiator desires a particular extension or context establishment flag (either critical or non-
critical) it sets the appropriate checksum flag. The context acceptor MUST ignore unsupported non-critical extensions or flags in the initiator’s context token (i.e., acceptors MUST NOT return an error just because there were unsupported non-critical extensions or flags in the initiator’s token). The acceptor MUST return GSS_S_UNAVAILABLE [RFC-2743] if there are unsupported critical extensions or flags in the initiator’s context token.

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

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_C_DELEG_FLAG</td>
<td>1</td>
</tr>
<tr>
<td>GSS_C_MUTUAL_FLAG</td>
<td>2</td>
</tr>
<tr>
<td>GSS_C_REPLAY_FLAG</td>
<td>4</td>
</tr>
<tr>
<td>GSS_C_SEQUENCE_FLAG</td>
<td>8</td>
</tr>
<tr>
<td>GSS_C_CONF_FLAG</td>
<td>16</td>
</tr>
<tr>
<td>GSS_C_INTEG_FLAG</td>
<td>32</td>
</tr>
<tr>
<td>GSS_C_ANON_FLAG</td>
<td>64</td>
</tr>
</tbody>
</table>

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(). An implementation that supports a particular extension SHOULD then set the appropriate flag in the checksum Flags field.

All existing context establishment flags are non-critical, and it is possible that a new context establishment flag can be added as a critical flag.

7.1.2. Channel Binding Information

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

1. Each integer field shall be formatted into four bytes, using little-endian byte ordering, for purposes of MD5 hash computation.

2. All input length fields within gss_buffer_desc [RFC-2744] elements of a gss_channel_bindings_struct [RFC-2744], 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 bindings structure, the Bnd field shall be set to 16 zero-valued bytes.

[Nicolas suggested that the only change that might be needed here was the use of SHA1 instead of MD5]

8. Per-Message and Context Deletion Tokens

The new per-message and context deletion token formats defined in
this document are designed to accommodate the requirements of newer crypto systems. The token layouts have also been designed to facilitate scatter-gather and in-place encryption without incurring significant performance penalties for implementations that do not allow for either scatter-gather or in-place encryption.

The design along with the rationale behind it is discussed in detail in the following sections.

8.1. Sequence Number and Direction Indicator

The sequence number for any per-message or context deletion token is a 64 bit integer (expressed in big endian order). One separate flag is used as the direction-indicator as described in section 8.2. Both the sequence number and the direction-indicator are protected by the encryption and checksum procedures as specified in section 8.4.

8.2. Flags Field

The Flags field is a one-byte bit vector used to indicate a set of attributes. The meanings of the flags are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SentByAcceptor</td>
<td>When set, this flag indicates the sender is the context acceptor. When not set, it indicates the sender is the context initiator.</td>
</tr>
<tr>
<td>1</td>
<td>Sealed</td>
<td>When set in Wrap tokens, this flag indicates confidentiality is provided for. It MUST not be set in MIC and context deletion tokens.</td>
</tr>
</tbody>
</table>

The rest of available bits are reserved for future use.

8.3. EC Field

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

In Wrap tokens with confidentiality, the EC field is used to encode the size (in bytes) of the random filler, as described in section 8.4.

In Wrap tokens without confidentiality, the EC field is used to encode the size (in bytes) of the trailing checksum, as described in section 8.4.

When AES is used, the EC field contains the hex value 00 0C in Wrap tokens without confidentiality, and 00 00 in Wrap tokens with confidentiality.

8.4. Encryption and Checksum Operations
The encryption algorithms defined by the crypto profiles provide for integrity protection. Therefore no separate checksum is needed.

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

In Wrap tokens that provide for confidentiality, the "header" (the first 16 bytes of the Wrap token) is appended to the plaintext data before encryption. Random filler is inserted between the plaintext-data and the "header", and there SHALL NOT be crypto-system garbage added by the decryption operation. The resulting Wrap token is ("header" | encrypt(plaintext-data | random-filler | "header")), where encrypt() is the encryption operation (which provides for integrity protection) defined in the crypto profile [KCRYPTO].

[A note from the design team (Sam, Nicolas, Ken, JK and Larry): constraints need to be added to kcrypto to keep the header at the end of the decrypted data. Without these constraints, we might have the header prepended to the front of the data and encode an 8 byte length for the plaintext data, which is less efficient.

Constraints to be added: Given the length of any plaintext data, there should always exist the next (possibly larger) size for which, when padding the to-be-encrypted data to that size, there will be no cryptosystem garbage added, and the number of bytes needed to pad to the next size is no larger than 64K. This is a small addition to kcrypto and we will bring it up at the IETF last call for kcrypto]

In Wrap tokens that do not provide for confidentiality, the checksum is calculated over the plaintext data concatenated with the token header (the first 16 bytes of the Wrap token). The resulting Wrap token is ("header" | plaintext-data | get_mic(plaintext-data | "header")), where get_mic() is the checksum operation defined in the crypto profile [KCRYPTO].

For MIC tokens, the checksum is first calculated over the token header (the first 16 bytes of the MIC token) and then the to-be-signed plaintext data.

For context deletion tokens, the checksum is calculated over the token header (the first 16 bytes of the context deletion token).

When AES is used, the checksum algorithm is HMAC_SHA1_96 and the
checksum size is 12 bytes. Data is pre-pended with a 16 byte confounder before encryption, and no padding is needed.

8.5. 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] 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 bytes of the token header, is rotated to the right by "RRC" bytes. The net result is that "RRC" bytes 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 byte sequence.

The RRC field is expressed as a two-byte 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.

8.6. Message Layout for Per-message and Context Deletion Tokens

The new message layouts are as follows.

MIC Token:

<table>
<thead>
<tr>
<th>Byte no</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>TOK_ID</td>
<td>Identification field. Tokens emitted by GSS_GetMIC() contain the hex value 04 04 in this field.</td>
</tr>
<tr>
<td>2</td>
<td>Flags</td>
<td>Attributes field, as described in Section 8.2.</td>
</tr>
<tr>
<td>3..7</td>
<td>Filler</td>
<td>Contains 5 bytes of hex value FF.</td>
</tr>
<tr>
<td>8..15</td>
<td>SND_SEQ</td>
<td>Sequence number field in cleartext, in big endian order.</td>
</tr>
<tr>
<td>16..last</td>
<td>SGN_CKSUM</td>
<td>Checksum of byte 0..15 and the &quot;to-be-signed&quot; data, where the checksum algorithm is defined by the crypto profile for the session key or subkey.</td>
</tr>
</tbody>
</table>

The Filler field is included in the checksum calculation for simplicity. This is common to both MIC and context deletion token checksum calculations.

Wrap Token:
### Token Details

<table>
<thead>
<tr>
<th>Byte no</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>TOK_ID</td>
<td>Identification field._TOKEN_ID contains the hex value 05 04 in this field.</td>
</tr>
<tr>
<td>2</td>
<td>Flags</td>
<td>Attributes field, as described in Section 8.2.</td>
</tr>
<tr>
<td>3</td>
<td>Filler</td>
<td>Contains the hex value FF.</td>
</tr>
<tr>
<td>4..5</td>
<td>EC</td>
<td>Contains the &quot;extra count&quot; field, in big endian order as described in section 8.3.</td>
</tr>
<tr>
<td>6..7</td>
<td>RRC</td>
<td>Contains the &quot;right rotation count&quot; in big endian order, as described in section 8.5.</td>
</tr>
<tr>
<td>8..15</td>
<td>SND_SEQ</td>
<td>Sequence number field in cleartext, in big endian order.</td>
</tr>
<tr>
<td>16..last</td>
<td>Data</td>
<td>Encrypted data or (plaintext data + checksum), as described in section 8.4, where the encryption or checksum algorithm is defined by the crypto profile for the session key or subkey.</td>
</tr>
</tbody>
</table>

### Context Deletion Token:

<table>
<thead>
<tr>
<th>Byte no</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>TOK_ID</td>
<td>Identification field._TOKEN_ID contains the hex value 04 05 in this field.</td>
</tr>
<tr>
<td>2</td>
<td>Flags</td>
<td>Attributes field, as described in Section 8.2.</td>
</tr>
<tr>
<td>3..7</td>
<td>Filler</td>
<td>Contains 5 bytes of hex value FF.</td>
</tr>
<tr>
<td>8..15</td>
<td>SND_SEQ</td>
<td>Sequence number field in cleartext, in big endian order.</td>
</tr>
<tr>
<td>16..N</td>
<td>SGN_CKSUM</td>
<td>Checksum of byte 0..15, where the checksum algorithm is defined by the crypto profile for the session key or subkey.</td>
</tr>
</tbody>
</table>

### 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].

#### 9.1. 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.

9.1.1. 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-NAMES 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

9.1.2. Kerberos-specific-codes

GSS_KRB5_S_KG_CCACHE_NOMATCH
/* "Principal in credential cache does not match desired name" */
GSS_KRB5_S_KG_KEYTAB_NOMATCH
/* "No principal in keytab matches desired name" */
GSS_KRB5_S_KG_TGT_MISSING
/* "Credential cache has no TGT" */
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" */
9.2. Buffer Sizes

All implementations of this specification shall be capable of accepting buffers of at least 16K bytes as input to GSS_GetMIC(), GSS_VerifyMIC(), and GSS_Wrap(), and shall be capable of accepting the output_token generated by GSS_Wrap() for a 16K byte input buffer as input to GSS_Unwrap(). Support for larger buffer sizes is optional but recommended.

10. 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 [GSSAPI-KRB5] when the key type corresponds to "older" algorithms. An alternative approach might be to retry sending the message with the sign or seal algorithm explicitly defined as in [GSSAPI-KRB5]. However, this would require the use of a mechanism such as [RFC-2478] to securely negotiate the algorithm or the use of a band mechanism to choose appropriate algorithms. For this reason, it is RECOMMENDED that the new token formats defined in this document can be used only if both peers are known during context negotiation to support the new mechanism (either because of the use of "new" enctypes or because of the use of Kerberos V extensions).

11. Security Considerations

It is possible that the KDC returns a session-key type that is not supported by the GSSAPI implementation (either on the client and the server). In this case the implementation MUST not try to use the key with a supported cryptosystem. This will ensure that no security weaknesses arise due to the use of an inappropriate key with an encryption algorithm.

In addition, the security problem described in [3DES] arising from the use of a service implementation with a GSSAPI mechanism supporting only DES and a Kerberos mechanism supporting both DES and Triple DES is actually a more generic one. It arises whenever the GSSAPI implementation does not support a stronger cryptosystem supported by the Kerberos mechanism. 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.

12. Acknowledgments

The authors wish to acknowledge the contributions from the following individuals:

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

Sam Hartman and Ken Raeburn suggested the "floating trailer" idea.
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 also provided valuable inputs on this draft.

13. References

13.1. Normative References


13.2. Informative References
14. 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

"Copyright (C) The Internet Society (date). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns."