Network Working Group
Request for Comments: 1964
Category: Standards Track

J. Linn
OpenVision Technologies
June 1996

The Kerberos Version 5 GSS-API Mechanism

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

ABSTRACT

This specification defines protocols, procedures, and conventions to be employed by peers implementing the Generic Security Service Application Program Interface (as specified in RFCs 1508 and 1509) when using Kerberos Version 5 technology (as specified in RFC 1510).

ACKNOWLEDGMENTS

Much of the material in this memo is based on working documents drafted by John Wray of Digital Equipment Corporation and on discussions, implementation activities, and interoperability testing involving Marc Horowitz, Ted Ts’o, and John Wray. Particular thanks are due to each of these individuals for their contributions towards development and availability of GSS-API support within the Kerberos Version 5 code base.

1. Token Formats

This section discusses protocol-visible characteristics of the GSS-API mechanism to be implemented atop Kerberos V5 security technology per RFC-1508 and RFC-1510; it defines elements of protocol for interoperability and is independent of language bindings per RFC-1509.

Tokens transferred between GSS-API peers (for security context management and per-message protection purposes) are defined. 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 RFC-1510 rather than within this specification.

To support ongoing experimentation, testing, and evolution of the specification, the Kerberos V5 GSS-API mechanism as defined in this and any successor memos will be identified with the following Object Identifier, as defined in RFC-1510, until the specification is advanced to the level of Proposed Standard RFC:

{iso(1), org(3), dod(5), internet(1), security(5), kerberosv5(2)}

Upon advancement to the level of Proposed Standard RFC, the Kerberos V5 GSS-API mechanism will be identified by an Object Identifier having the value:

{iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) krb5(2)}

1.1. Context Establishment Tokens

Per RFC-1508, Appendix B, the initial context establishment token will be enclosed within framing as follows:

```
InitialContextToken ::=
[APPLICATION 0] IMPLICIT SEQUENCE {
        thisMech        MechType
                -- MechType is OBJECT IDENTIFIER
                -- representing "Kerberos V5"
        innerContextToken ANY DEFINED BY thisMech
                -- contents mechanism-specific;
                -- ASN.1 usage within innerContextToken
                -- is not required
        }
```

The innerContextToken of the initial context token will consist of a Kerberos V5 KRB_AP_REQ message, preceded by a two-byte token-id (TOK_ID) field, which shall contain the value 01 00.

The above GSS-API framing shall be applied to all tokens emitted by the Kerberos V5 GSS-API mechanism, including KRB_AP_REP, KRB_ERROR, context-deletion, and per-message tokens, not just to the initial token in a context establishment sequence. While not required by RFC-1508, this enables implementations to perform enhanced error-checking. The innerContextToken field of context establishment tokens for the Kerberos V5 GSS-API mechanism will contain a Kerberos message (KRB_AP_REQ, KRB_AP_REP or KRB_ERROR), preceded by a 2-byte TOK_ID field containing 01 00 for KRB_AP_REQ messages, 02 00 for KRB_AP_REP messages and 03 00 for KRB_ERROR messages.

1.1.1. Initial Token

Relevant KRB_AP_REQ syntax (from RFC-1510) is as follows:

```
   AP-REQ ::= [APPLICATION 14] SEQUENCE {
           pvno [0]        INTEGER,        -- indicates Version 5
           msg-type [1]    INTEGER,        -- indicates KRB_AP_REQ
           ap-options[2]   APOptions,
           ticket[3]       Ticket,
           authenticator[4]        EncryptedData
   }

   APOptions ::= BIT STRING {
           reserved (0),
           use-session-key (1),
           mutual-required (2)
   }

   Ticket ::= [APPLICATION 1] SEQUENCE {
           tkt-vno [0]     INTEGER,        -- indicates Version 5
           realm [1]       Realm,
           sname [2]       PrincipalName,
           enc-part [3]    EncryptedData
   }

   -- Encrypted part of ticket
   EncTicketPart ::= [APPLICATION 3] SEQUENCE {
           flags[0]        TicketFlags,
           key[1]          EncryptionKey,
           crealm[2]       Realm,
           cname[3]        PrincipalName,
           transited[4]    TransitedEncoding,
           authtime[5]     KerberosTime,
           starttime[6]    KerberosTime OPTIONAL,
           endtime[7]      KerberosTime,
           renew-till[8]   KerberosTime OPTIONAL,
           caddr[9]        HostAddresses OPTIONAL,
           authorization-data[10]  AuthorizationData OPTIONAL
   }

   -- Unencrypted authenticator
   Authenticator ::= [APPLICATION 2] SEQUENCE  {
           authenticator-vno[0]    INTEGER,
           crealm[1]               Realm,
           cname[2]                PrincipalName,
           cksum[3]                Checksum OPTIONAL,
           cusec[4]                INTEGER,
           ctime[5]                KerberosTime,
           subkey[6]               EncryptionKey OPTIONAL,
           seq-number[7]           INTEGER OPTIONAL,
           authorization-data[8]   AuthorizationData OPTIONAL
   }
```

For purposes of this specification, the authenticator shall include the optional sequence number, and the checksum field shall be used to convey channel binding, service flags, and optional delegation information. The checksum will have a type of 0x8003 (a value being registered within the Kerberos protocol specification), and a value field of at least 24 bytes in length. The length of the value field is extended beyond 24 bytes if and only if an optional facility to carry a Kerberos-defined KRB_CRED message for delegation purposes is supported by an implementation and active on a context. When delegation is active, a TGT with its FORWARDABLE flag set will be transferred within the KRB_CRED message.

The checksum value field's format is as follows:

Byte	Name	Description
0..3	Lgth	Number of bytes in Bnd field; Currently contains hex 10 00 00 00 (16, represented in little-endian form)
4..19	Bnd	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.
20..23	Flags	Bit vector of context-establishment flags, with values consistent with RFC-1509, p. 41: 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 The resulting bit vector is encoded into bytes 20..23 in little-endian form.
24..25	DlgOpt	The Delegation Option identifier (=1) [optional]
26..27	Dlgth	The length of the Deleg field. [optional]
28..n	Deleg	A KRB_CRED message (n = Dlgth + 29) [optional]

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 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 bindings structure, the Bnd field shall be set to 16 zero-valued bytes.

In the initial Kerberos V5 GSS-API mechanism token (KRB_AP_REQ token) from initiator to target, the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, and GSS_C_SEQUENCE_FLAG values shall each be set as the logical AND of the initiator’s corresponding request flag to GSS_Init_sec_context() and a Boolean indicator of whether that optional service is available to GSS_Init_sec_context()’s caller. GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG, for which no corresponding context-level input indicator flags to GSS_Init_sec_context() exist, shall each be set to indicate whether their respective per-message protection services are available for use on the context being established.

When input source address channel binding values are provided by a caller (i.e., unless the input argument is GSS_C_NO_BINDINGS or the source address specifier value within the input structure is GSS_C_NULL_ADDRTYPE), and the corresponding token received from the context’s peer bears address restrictions, it is recommended that an implementation of the Kerberos V5 GSS-API mechanism should check that the source address as provided by the caller matches that in the received token, and should return the GSS_S_BAD_BINDINGS major_status value if a mismatch is detected. Note: discussion is ongoing about the strength of recommendation to be made in this area, and on the circumstances under which such a recommendation should be applicable; implementors are therefore advised that changes on this matter may be included in subsequent versions of this specification.

1.1.2. Response Tokens

A context establishment sequence based on the Kerberos V5 mechanism will perform one-way authentication (without confirmation or any return token from target to initiator in response to the initiator’s KRB_AP_REQ) if the mutual_req bit is not set in the application’s call to GSS_Init_sec_context(). Applications requiring confirmation that their authentication was successful should request mutual authentication, resulting in a "mutual-required" indication within KRB_AP_REQ APoptions and the setting of the mutual_req bit in the flags field of the authenticator checksum. In response to such a request, the context target will reply to the initiator with a token containing either a KRB_AP_REP or KRB_ERROR, completing the mutual context establishment exchange.

Relevant KRB_AP_REP syntax is as follows:

```
   AP-REP ::= [APPLICATION 15] SEQUENCE {
           pvno [0]        INTEGER,        -- represents Kerberos V5
           msg-type [1]    INTEGER,        -- represents KRB_AP_REP
           enc-part [2]    EncryptedData
   }

   EncAPRepPart ::= [APPLICATION 27] SEQUENCE {
           ctime [0]       KerberosTime,
           cusec [1]       INTEGER,
           subkey [2]      EncryptionKey OPTIONAL,
           seq-number [3]  INTEGER OPTIONAL
   }
```

The optional seq-number element within the AP-REP’s EncAPRepPart shall be included.

The syntax of KRB_ERROR is as follows:

```
   KRB-ERROR ::=   [APPLICATION 30] SEQUENCE {
           pvno[0]         INTEGER,
           msg-type[1]     INTEGER,
           ctime[2]        KerberosTime OPTIONAL,
           cusec[3]        INTEGER OPTIONAL,
           stime[4]        KerberosTime,
           susec[5]        INTEGER,
           error-code[6]   INTEGER,
           crealm[7]       Realm OPTIONAL,
           cname[8]        PrincipalName OPTIONAL,
           realm[9]        Realm, -- Correct realm
           sname[10]       PrincipalName, -- Correct name
           e-text[11]      GeneralString OPTIONAL,
           e-data[12]      OCTET STRING OPTIONAL
   }
```

Values to be transferred in the error-code field of a KRB-ERROR message are defined in [RFC-1510], not in this specification.

1.2. Per-Message and Context Deletion Tokens

Three classes of tokens are defined in this section: "MIC" tokens, emitted by calls to GSS_GetMIC() (formerly GSS_Sign()) and consumed by calls to GSS_VerifyMIC() (formerly GSS_Verify()), "Wrap" tokens, emitted by calls to GSS_Wrap() (formerly GSS_Seal()) and consumed by calls to GSS_Unwrap() (formerly GSS_Unseal()), and context deletion tokens, emitted by calls to GSS_Delete_sec_context() and consumed by calls to GSS_Process_context_token(). Note: References to GSS-API per-message routines in the remainder of this specification will be based on those routines’ newer recommended names rather than those names’ predecessors.

Several variants of cryptographic keys are used in generation and processing of per-message tokens:

(1) context key: uses Kerberos session key (or subkey, if present in authenticator emitted by context initiator) directly

(2) confidentiality key: forms variant of context key by exclusive-OR with the hexadecimal constant f0f0f0f0f0f0f0f0.

(3) MD2.5 seed key: forms variant of context key by reversing the bytes of the context key (i.e. if the original key is the 8-byte sequence {aa, bb, cc, dd, ee, ff, gg, hh}, the seed key will be {hh, gg, ff, ee, dd, cc, bb, aa}).

1.2.1. Per-message Tokens - MIC

Use of the GSS_GetMIC() call yields a token, 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:

Byte no	Name	Description
0..1	TOK_ID	Identification field. Tokens emitted by GSS_GetMIC() contain the hex value 01 01 in this field.
2..3	SGN_ALG	Integrity algorithm indicator. 00 00 - DES MAC MD5 01 00 - MD2.5 02 00 - DES MAC
4..7	Filler	Contains ff ff ff ff
8..15	SND_SEQ	Sequence number field.
16..23	SGN_CKSUM	Checksum of "to-be-signed data", calculated according to algorithm specified in SGN_ALG field.

GSS-API tokens must be encapsulated within the higher-level protocol by the application; no embedded length field is necessary.

1.2.1.1. Checksum

Checksum calculation procedure (common to all algorithms): Checksums are calculated over the data field, logically prepended by the first 8 bytes of the plaintext packet header. The resulting value binds the data to the packet type and signature algorithm identifier fields.

DES MAC MD5 algorithm: The checksum is formed by computing an MD5 [RFC-1321] hash over the plaintext data, and then computing a DES-CBC MAC on the 16-byte MD5 result. A standard 64-bit DES-CBC MAC is computed per [FIPS-PUB-113], employing the context key and a zero IV. The 8-byte result is stored in the SGN_CKSUM field.

MD2.5 algorithm: The checksum is formed by first DES-CBC encrypting a 16-byte zero-block, using a zero IV and a key formed by reversing the bytes of the context key (i.e. if the original key is the 8-byte sequence {aa, bb, cc, dd, ee, ff, gg, hh}, the checksum key will be {hh, gg, ff, ee, dd, cc, bb, aa}). The resulting 16-byte value is logically prepended to the to-be-signed data. A standard MD5 checksum is calculated over the combined data, and the first 8 bytes of the result are stored in the SGN_CKSUM field. Note 1: we refer to this algorithm informally as "MD2.5" to connote the fact that it uses half of the 128 bits generated by MD5; use of only a subset of the MD5 bits is intended to protect against the prospect that data could be postfixed to an existing message with corresponding modifications being made to the checksum. Note 2: This algorithm is fairly novel and has received more limited evaluation than that to which other integrity algorithms have been subjected. An initial, limited evaluation indicates that it may be significantly weaker than DES MAC MD5.

DES-MAC algorithm: A standard 64-bit DES-CBC MAC is computed on the plaintext data per [FIPS-PUB-113], employing the context key and a zero IV. Padding procedures to accomodate plaintext data lengths which may not be integral multiples of 8 bytes are defined in [FIPS-PUB-113]. The result is an 8-byte value, which is stored in the SGN_CKSUM field. Support for this algorithm may not be present in all implementations.

1.2.1.2. Sequence Number

Sequence number field: The 8 byte plaintext sequence number field is formed from the sender's four-byte sequence number as follows. If the four bytes of the sender's sequence number are named s0, s1, s2 and s3 (from least to most significant), the plaintext sequence number field is the 8 byte sequence: (s0, s1, s2, s3, di, di, di, di), where 'di' is the direction-indicator (Hex 0 - sender is the context initiator, Hex FF - sender is the context acceptor). The field is then DES-CBC encrypted using the context key and an IV formed from the first 8 bytes of the previously calculated SGN_CKSUM field. After sending a GSS_GetMIC() or GSS_Wrap() token, the sender's sequence number is incremented by one.

The receiver of the token will first verify the SGN_CKSUM field. If valid, the sequence number field may be decrypted and compared to the expected sequence number. The repetition of the (effectively 1-bit) direction indicator within the sequence number field provides redundancy so that the receiver may verify that the decryption succeeded.

Since the checksum computation is used as an IV to the sequence number decryption, attempts to splice a checksum and sequence number from different messages will be detected. The direction indicator will detect packets that have been maliciously reflected.

The sequence number provides a basis for detection of replayed tokens. Replay detection can be performed using state information retained on received sequence numbers, interpreted in conjunction with the security context on which they arrive.

Provision of per-message replay and out-of-sequence detection services is optional for implementations of the Kerberos V5 GSS-API mechanism. Further, it is recommended that implementations of the Kerberos V5 GSS-API mechanism which offer these services should honor a caller's request that the services be disabled on a context. Specifically, if replay_det_req_flag is input FALSE, replay_det_state should be returned FALSE and the GSS_DUPLICATE_TOKEN and GSS_OLD_TOKEN stati should not be indicated as a result of duplicate detection when tokens are processed; if sequence_req_flag is input FALSE, sequence_state should be returned FALSE and GSS_DUPLICATE_TOKEN, GSS_OLD_TOKEN, and GSS_UNSEQ_TOKEN stati should not be indicated as a result of out-of-sequence detection when tokens are processed.

1.2.2. Per-message Tokens - Wrap

Use of the GSS_Wrap() call yields a token which encapsulates the input user data (optionally encrypted) along with associated integrity check quantities. The token emitted by GSS_Wrap() consists of an integrity header whose format is identical to that emitted by GSS_GetMIC() (except that the TOK_ID field contains the value 02 01), followed by a body portion that contains either the plaintext data (if SEAL_ALG = ff ff) or encrypted data for any other supported value of SEAL_ALG. Currently, only SEAL_ALG = 00 00 is supported, and means that DES-CBC encryption is being used to protect the data.

The GSS_Wrap() token has the following format:

Byte no	Name	Description
0..1	TOK_ID	Identification field. Tokens emitted by GSS_Wrap() contain the hex value 02 01 in this field.
2..3	SGN_ALG	Checksum algorithm indicator. 00 00 - DES MAC MD5 01 00 - MD2.5 02 00 - DES MAC
4..5	SEAL_ALG	ff ff - none 00 00 - DES
6..7	Filler	Contains ff ff
8..15	SND_SEQ	Encrypted sequence number field.
16..23	SGN_CKSUM	Checksum of plaintext padded data, calculated according to algorithm specified in SGN_ALG field.
24..last	Data	encrypted or plaintext padded data

GSS-API tokens must be encapsulated within the higher-level protocol by the application; no embedded length field is necessary.

1.2.2.1. Checksum

Checksum calculation procedure (common to all algorithms): Checksums are calculated over the plaintext padded data field, logically prepended by the first 8 bytes of the plaintext packet header. The resulting signature binds the data to the packet type, protocol version, and signature algorithm identifier fields.

DES MAC MD5 algorithm: The checksum is formed by computing an MD5 hash over the plaintext padded data, and then computing a DES-CBC MAC on the 16-byte MD5 result. A standard 64-bit DES-CBC MAC is computed per [FIPS-PUB-113], employing the context key and a zero IV. The 8-byte result is stored in the SGN_CKSUM field.

MD2.5 algorithm: The checksum is formed by first DES-CBC encrypting a 16-byte zero-block, using a zero IV and a key formed by reversing the bytes of the context key (i.e., if the original key is the 8-byte sequence {aa, bb, cc, dd, ee, ff, gg, hh}, the checksum key will be {hh, gg, ff, ee, dd, cc, bb, aa}). The resulting 16-byte value is logically pre-pended to the "to-be-signed data". A standard MD5 checksum is calculated over the combined data, and the first 8 bytes of the result are stored in the SGN_CKSUM field.

DES-MAC algorithm: A standard 64-bit DES-CBC MAC is computed on the plaintext padded data per [FIPS-PUB-113], employing the context key and a zero IV. The plaintext padded data is already assured to be an integral multiple of 8 bytes; no additional padding is required or applied in order to accomplish MAC calculation. The result is an 8-byte value, which is stored in the SGN_CKSUM field. Support for this lgorithm may not be present in all implementations.

1.2.2.2. Sequence Number

Sequence number field: The 8 byte plaintext sequence number field is formed from the sender’s four-byte sequence number as follows. If the four bytes of the sender’s sequence number are named s0, s1, s2 and s3 (from least to most significant), the plaintext sequence number field is the 8 byte sequence: (s0, s1, s2, s3, di, di, di, di), where ’di’ is the direction-indicator (Hex 0 - sender is the context initiator, Hex FF - sender is the context acceptor).

The field is then DES-CBC encrypted using the context key and an IV formed from the first 8 bytes of the SEAL_CKSUM field.

After sending a GSS_GetMIC() or GSS_Wrap() token, the sender’s sequence numbers are incremented by one.

1.2.2.3. Padding

Data padding: Before encryption and/or signature calculation, plaintext data is padded to the next highest multiple of 8 bytes, by appending between 1 and 8 bytes, the value of each such byte being the total number of pad bytes. For example, given data of length 20 bytes, four pad bytes will be appended, and each byte will contain the hex value 04. An 8-byte random confounder is prepended to the data, and signatures are calculated over the resulting padded plaintext.

After padding, the data is encrypted according to the algorithm specified in the SEAL_ALG field. For SEAL_ALG=DES (the only non-null algorithm currently supported), the data is encrypted using DES-CBC, with an IV of zero. The key used is derived from the established context key by XOR-ing the context key with the hexadecimal constant f0f0f0f0f0f0f0f0.

1.2.3. Context deletion token

The token emitted by GSS_Delete_sec_context() is based on the packet format for tokens emitted by GSS_GetMIC(). The context-deletion token has the following format:

Byte no	Name	Description
0..1	TOK_ID	Identification field. Tokens emitted by GSS_Delete_sec_context() contain the hex value 01 02 in this field.
2..3	SGN_ALG	Integrity algorithm indicator. 00 00 - DES MAC MD5 01 00 - MD2.5 02 00 - DES MAC
4..7	Filler	Contains ff ff ff ff
8..15	SND_SEQ	Sequence number field.
16..23	SGN_CKSUM	Checksum of "to-be-signed data", calculated according to algorithm specified in SGN_ALG field.

SGN_ALG and SND_SEQ will be calculated as for tokens emitted by GSS_GetMIC(). The SGN_CKSUM will be calculated as for tokens emitted by GSS_GetMIC(), except that the user-data component of the "to-be-signed" data will be a zero-length string.

2. Name Types and Object Identifiers

This section discusses the name types which may be passed as input to the Kerberos V5 GSS-API mechanism's GSS_Import_name() call, and their associated identifier values. It defines interface elements in support of portability, and assumes use of C language bindings per RFC-1509. In addition to specifying OID values for name type identifiers, symbolic names are included and recommended to GSS-API implementors in the interests of convenience to callers. It is understood that not all implementations of the Kerberos V5 GSS-API mechanism need support all name types in this list, and that additional name forms will likely be added to this list over time. Further, the definitions of some or all name types may later migrate to other, mechanism-independent, specifications. The occurrence of a name type in this specification is specifically not intended to suggest that the type may be supported only by an implementation of the Kerberos V5 mechanism. In particular, the occurrence of the string "_KRB5_" in the symbolic name strings constitutes a means to unambiguously register the name strings, avoiding collision with other documents; it is not meant to limit the name types' usage or applicability.

For purposes of clarification to GSS-API implementors, this section's discussion of some name forms describes means through which those forms can be supported with existing Kerberos technology. These discussions are not intended to preclude alternative implementation strategies for support of the name forms within Kerberos mechanisms or mechanisms based on other technologies. To enhance application portability, implementors of mechanisms are encouraged to support name forms as defined in this section, even if their mechanisms are independent of Kerberos V5.

2.1. Mandatory Name Forms

This section discusses name forms which are to be supported by all conformant implementations of the Kerberos V5 GSS-API mechanism.

2.1.1. Kerberos Principal Name Form

This name form shall be represented by the Object Identifier {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) krb5(2) krb5_name(1)}. The recommended symbolic name for this type is "GSS_KRB5_NT_PRINCIPAL_NAME".

This name type corresponds to the single-string representation of a Kerberos name. (Within the MIT Kerberos V5 implementation, such names are parseable with the krb5_parse_name() function.) The elements included within this name representation are as follows, proceeding from the beginning of the string:

(1) One or more principal name components; if more than one principal name component is included, the components are separated by ‘/‘. Arbitrary octets may be included within principal name components, with the following constraints and special considerations:

(1a) Any occurrence of the characters ‘@‘ or ‘/‘ within a name component must be immediately preceded by the ‘\‘ quoting character, to prevent interpretation as a component or realm separator.

(1b) The ASCII newline, tab, backspace, and null characters may occur directly within the component or may be represented, respectively, by ‘\n‘, ‘\t‘, ‘\b‘, or ‘\0‘.

(1c) If the ‘\‘ quoting character occurs outside the contexts described in (1a) and (1b) above, the following character is interpreted literally. As a special case, this allows the doubled representation ‘\\‘ to represent a single occurrence of the quoting character.

(1d) An occurrence of the ‘\‘ quoting character as the last character of a component is illegal.

(2) Optionally, a ‘@‘ character, signifying that a realm name immediately follows. If no realm name element is included, the local realm name is assumed. The ‘/‘ , ‘:‘, and null characters may not occur within a realm name; the ‘@‘, newline, tab, and backspace characters may be included using the quoting conventions described in (1a), (1b), and (1c) above.

2.1.2. Host-Based Service Name Form

This name form has been incorporated at the mechanism-independent GSS-API level as of GSS-API, Version 2. This subsection retains the Object Identifier and symbolic name assignments previously made at the Kerberos V5 GSS-API mechanism level, and adopts the definition as promoted to the mechanism-independent level.

This name form shall be represented by the Object Identifier {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) service_name(4)}. The previously recommended symbolic name for this type is "GSS_KRB5_NT_HOSTBASED_SERVICE_NAME". The currently preferred symbolic name for this type is "GSS_C_NT_HOSTBASED_SERVICE".

This name type is used to represent services associated with host computers. This name form is constructed using two elements, "service" and "hostname", as follows:

```
service@hostname
```

When a reference to a name of this type is resolved, the "hostname" is canonicalized by attempting a DNS lookup and using the fully-qualified domain name which is returned, or by using the "hostname" as provided if the DNS lookup fails. The canonicalization operation also maps the host’s name into lower-case characters.

The "hostname" element may be omitted. If no "@" separator is included, the entire name is interpreted as the service specifier, with the "hostname" defaulted to the canonicalized name of the local host.

Values for the "service" element will be registered with the IANA.

2.1.3. Exported Name Object Form for Kerberos V5 Mechanism

Support for this name form is not required for GSS-V1 implementations, but will be required for use in conjunction with the GSS_Export_name() call planned for GSS-API Version 2. Use of this name form will be signified by a "GSS-API Exported Name Object" OID value which will be defined at the mechanism-independent level for

GSS-API Version 2.

This name type represents a self-describing object, whose framing structure will be defined at the mechanism-independent level for GSS-API Version 2. When generated by the Kerberos V5 mechanism, the Mechanism OID within the exportable name shall be that of the Kerberos V5 mechanism. The name component within the exportable name shall be a contiguous string with structure as defined for the Kerberos Principal Name Form.

In order to achieve a distinguished encoding for comparison purposes, the following additional constraints are imposed on the export operation:

(1) all occurrences of the characters ‘@‘, ‘/‘, and ‘\‘ within principal components or realm names shall be quoted with an immediately-preceding ‘\‘.

(2) all occurrences of the null, backspace, tab, or newline characters within principal components or realm names will be represented, respectively, with ‘\0‘, ‘\b‘, ‘\t‘, or ‘\n‘.

(3) the ‘\‘ quoting character shall not be emitted within an exported name except to accomodate cases (1) and (2).

2.2. Optional Name Forms

This section discusses additional name forms which may optionally be supported by implementations of the Kerberos V5 GSS-API mechanism. It is recognized that some of the name forms cited here are derived from UNIX(tm) operating system platforms; some listed forms may be irrelevant to non-UNIX platforms, and definition of additional forms corresponding to such platforms may also be appropriate. It is also recognized that OS-specific functions outside GSS-API are likely to exist in order to perform translations among these forms, and that GSS-API implementations supporting these forms may themselves be layered atop such OS-specific functions. Inclusion of this support within GSS-API implementations is intended as a convenience to applications.

2.2.1. User Name Form

This name form shall be represented by the Object Identifier {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) user_name(1)}. The recommended symbolic name for this type is "GSS_KRB5_NT_USER_NAME".

This name type is used to indicate a named user on a local system.

Its interpretation is OS-specific. This name form is constructed as:

```
username
```

Assuming that users’ principal names are the same as their local operating system names, an implementation of GSS_Import_name() based on Kerberos V5 technology can process names of this form by postfixing an "@" sign and the name of the local realm.

2.2.2. Machine UID Form

This name form shall be represented by the Object Identifier {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) machine_uid_name(2)}. The recommended symbolic name for this type is "GSS_KRB5_NT_MACHINE_UID_NAME".

This name type is used to indicate a numeric user identifier corresponding to a user on a local system. Its interpretation is OS-specific. The gss_buffer_desc representing a name of this type should contain a locally-significant uid_t, represented in host byte order. The GSS_Import_name() operation resolves this uid into a username, which is then treated as the User Name Form.

2.2.3. String UID Form

This name form shall be represented by the Object Identifier {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) string_uid_name(3)}. The recommended symbolic name for this type is "GSS_KRB5_NT_STRING_UID_NAME".

This name type is used to indicate a string of digits representing the numeric user identifier of a user on a local system. Its interpretation is OS-specific. This name type is similar to the Machine UID Form, except that the buffer contains a string representing the uid_t.

3. Credentials Management

The Kerberos V5 protocol uses different credentials (in the GSSAPI sense) for initiating and accepting security contexts. Normal clients receive a ticket-granting ticket (TGT) and an associated session key at "login" time; the pair of a TGT and its corresponding session key forms a credential which is suitable for initiating security contexts. A ticket-granting ticket, its session key, and any other (ticket, key) pairs obtained through use of the ticket-granting-ticket, are typically stored in a Kerberos V5 credentials cache, sometimes known as a ticket file.

The encryption key used by the Kerberos server to seal tickets for a particular application service forms the credentials suitable for accepting security contexts. These service keys are typically stored in a Kerberos V5 key table, or srvtab file. In addition to their use as accepting credentials, these service keys may also be used to obtain initiating credentials for their service principal.

The Kerberos V5 mechanism’s credential handle may contain references to either or both types of credentials. It is a local matter how the Kerberos V5 mechanism implementation finds the appropriate Kerberos V5 credentials cache or key table.

However, when the Kerberos V5 mechanism attempts to obtain initiating credentials for a service principal which are not available in a credentials cache, and the key for that service principal is available in a Kerberos V5 key table, the mechanism should use the service key to obtain initiating credentials for that service. This should be accomplished by requesting a ticket-granting-ticket from the Kerberos Key Distribution Center (KDC), and decrypting the KDC’s reply using the service key.

4. 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-1509.

4.1. 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 implementors 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.

4.1.1. 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" */
```

4.1.2. 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" */
```

4.2. Quality of Protection Values

This section defines Quality of Protection (QOP) values to be used with the Kerberos V5 GSS-API mechanism as input to GSS_Wrap() and GSS_GetMIC() routines in order to select among alternate integrity and confidentiality algorithms. Additional QOP values may be added in future versions of this specification. Non-overlapping bit positions are and will be employed in order that both integrity and confidentiality QOP may be selected within a single parameter, via inclusive-OR of the specified integrity and confidentiality values.

4.2.1. Integrity Algorithms

The following Quality of Protection (QOP) values are currently defined for the Kerberos V5 GSS-API mechanism, and are used to select among alternate integrity checking algorithms.

```
GSS_KRB5_INTEG_C_QOP_MD5        (numeric value: 1)
        /* Integrity using partial MD5 ("MD2.5") of plaintext */

GSS_KRB5_INTEG_C_QOP_DES_MD5    (numeric value: 2)
        /* Integrity using DES MAC of MD5 of plaintext */

GSS_KRB5_INTEG_C_QOP_DES_MAC    (numeric value: 3)
        /* Integrity using DES MAC of plaintext */
```

4.2.2. Confidentiality Algorithms

Only one confidentiality QOP value is currently defined for the Kerberos V5 GSS-API mechanism:

```
GSS_KRB5_CONF_C_QOP_DES         (numeric value: 0)
        /* Confidentiality with DES */
```

Note: confidentiality QOP should be indicated only by GSS-API calls capable of providing confidentiality services. If non-zero confidentiality QOP values are defined in future to represent different algorithms, therefore, the bit positions containing those values should be cleared before being returned by implementations of GSS_GetMIC() and GSS_VerifyMIC().

4.3. Buffer Sizes

All implementations of this specification shall be capable of accepting buffers of at least 16 Kbytes 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 16 Kbyte input buffer as input to GSS_Unwrap(). Support for larger buffer sizes is optional but recommended.

5. Security Considerations

Security issues are discussed throughout this memo.

6. References

[RFC-1321]: Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC-1508]: Linn, J., "Generic Security Service Application Program Interface", RFC 1508, September 1993.

[RFC-1509]: Wray, J., "Generic Security Service Application Program Interface: C-bindings", RFC 1509, September 1993.

[RFC-1510]: Kohl, J., and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[FIPS-PUB-113]: National Bureau of Standards, Federal Information Processing Standard 113, "Computer Data Authentication", May 1985.

AUTHOR’S ADDRESS

John Linn
OpenVision Technologies
One Main St.
Cambridge, MA 02142 USA

Phone: +1 617.374.2245
EMail: John.Linn@ov.com