Internet-Draft
Common Authentication Technology WG
draft-ietf-cat-krb5gss-mech2-00.txt

Tom Yu
MIT

7 August 1998

The Kerberos Version 5 GSSAPI Mechanism, Version 2

STATUS OF THIS MEMO

Comments on this document should be sent to "ietf-cat-wg@lists.stanford.edu", the IETF Common Authentication Technology WG discussion list.

ABSTRACT

This specification defines protocols, procedures, and conventions to be employed by peers implementing the Generic Security Service Application Program Interface (as specified in RFC 2078) when using Kerberos Version 5 technology (as specified in RFC 1510). This obsoletes RFC 1964. [XXX not quite yet because it's still missing some sections]

ACKNOWLEDGEMENTS

Much of the material in this specification is based on work done for Cygnus Solutions by Marc Horowitz.

1. Introduction

The previous GSSAPI Kerberos V5 mechanism, described in RFC 1964, has several flaws which make integrating new encryption types (enctypes) more difficult. The context establishment token format requires that the authenticator of AP-REQ messages contain a cleartext data structure in its checksum field, which is a needless and potentially confusing overloading of that field. The number assignments for checksum algorithms are also inconsistent between the Kerberos protocol and the GSSAPI mechanism. The previous mechanism also assumes that both checksums and cryptosystem blocksizes are eight bytes. There are no

backward-compatible ways to remedy these shortcomings.

Defining all GSSAPI tokens for the new Kerberos V5 mechanism in terms of the Kerberos protocol specification ensures that new encryption types and checksum types may be automatically used as they are defined for the Kerberos protocol.

2. Token Formats

All tokens, not just the initial token, are framed as the InitialContextToken described in RFC 2078 section 3.1. The innerContextToken element of the token will not itself be encoded in ASN.1. The rationale for not using ASN.1 in the inner token is that some implementors may wish to implement this mechanism in a kernel or other similarly constrained application where ASN.1 encoding may be cumbersome. Also, ASN.1 encoders and decoders are very difficult to implement completely correctly, so keeping ASN.1 usage to a minimum decreases the probability of bugs in the implementation of the mechanism.

All integer fields are in network byte order. All other fields have the fixed size shown.

2.1. Packet Notation

The order of transmission of this protocol is described at the octet level. Packet diagrams depict bits in the order of transmission, assuming that individual octets are transmitted with the most significant bit (MSB) first. The diagrams read from left to right and from top to bottom, as in printed English. When bits are numbered, they are numbered in the order of transmission, not the order of the significance of the bits; thus, bit number 0 is the MSB of the first octet. Numbers prefixed by the string "0x" are in hexadecimal notation, as in the C programming language. Fixed-length fields that appear to be aligned on 32-bit boundaries are not necessarily aligned if they follow a variable length field. No padding should be used to force alignment of such fields to a 32-bit boundary.

2.2. Mechanism OID

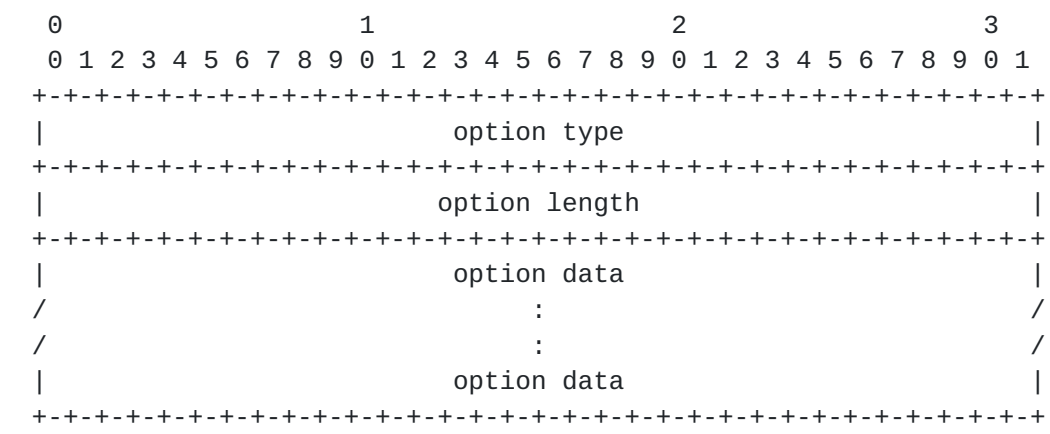
The Object Identifier (OID) of the new krb5 v2 mechanism is:

{iso(1) member-body(2) US(840) mit(113554) infosys(1) gssapi(2) krb5v2(3)}

2.3. Context Establishment

2.3.1. Option Format

Options contained in context establishment tokens shall have the following format:



option type (32 bits)
The type identifier of the following option.

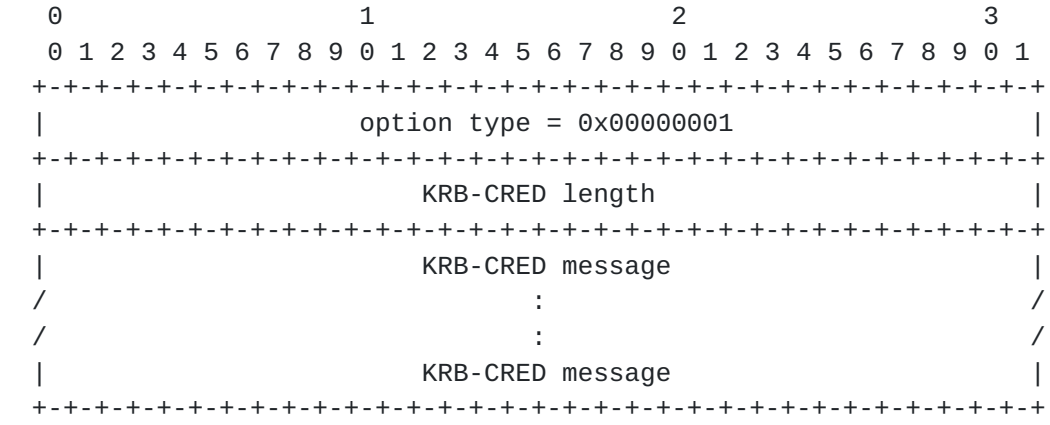
option length (32 bits)
The length in bytes of the following option.

option data (variable length)
The actual option data.

Any number of options may appear in an initator or acceptor token. The final option in a token must be the null option, in order to mark the end of the list.

2.3.1.1. Delegated Credentials Option

Only the initiator may use this option. The format of the delegated credentials option is as follows:



option type (32 bits)
The option type for this option shall be 0x00000001.

KRB-CRED length (32 bits)

The length in bytes of the following KRB-CRED message.

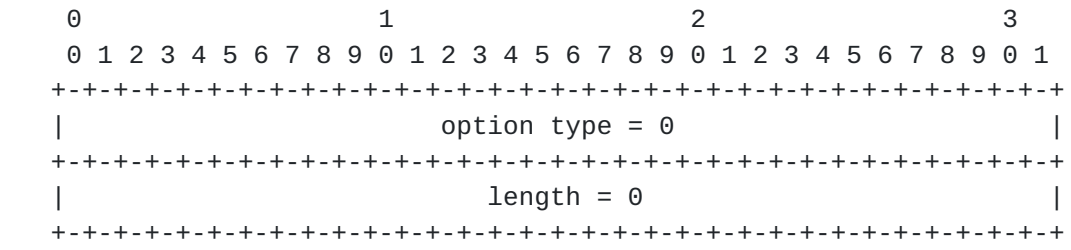
KRB-CRED message (variable length)

The option data for this option shall be the KRB-CRED message that

contains the credentials being delegated (forwarded) to the context acceptor. Only the initiator may use this option.

2.3.1.2. Null Option

The Null option terminates the option list:

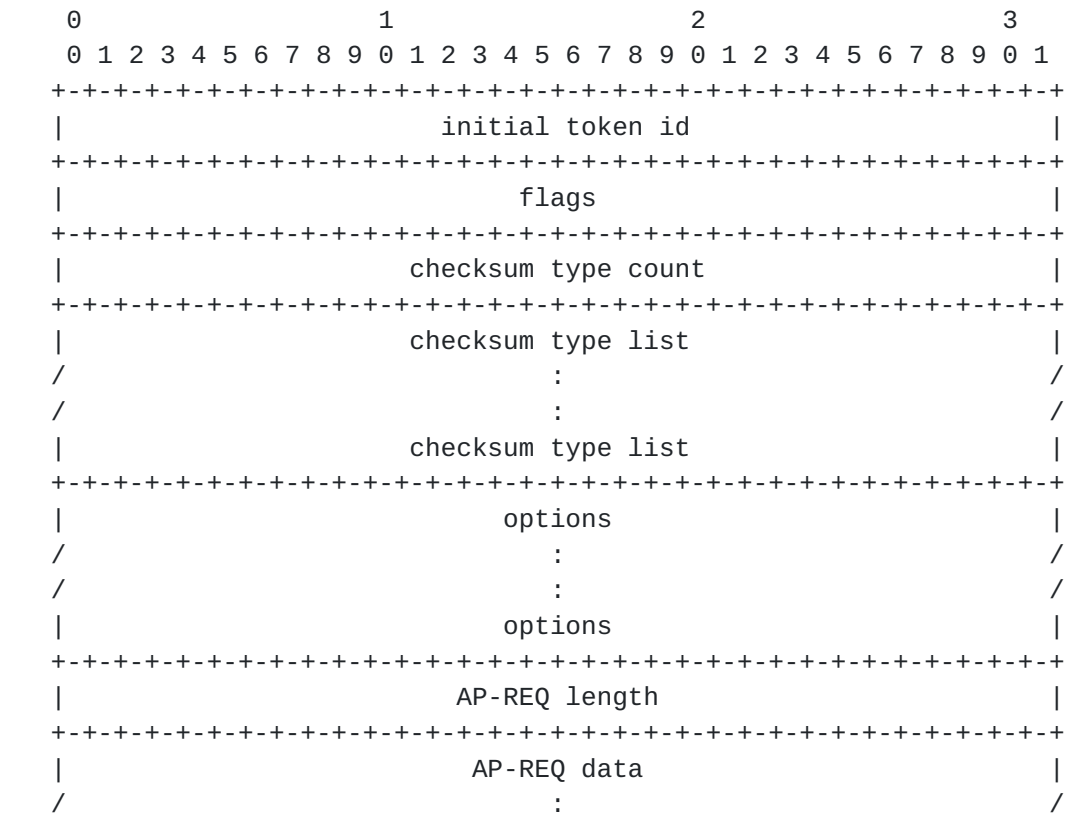


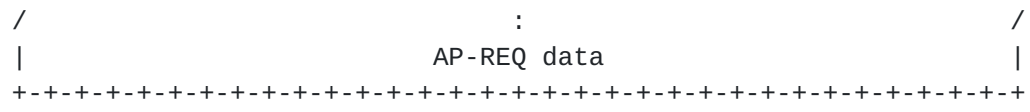
option type (32 bits)
The option type of this option must be zero.

option length (32 bits)
The length of this option must be zero.

2.3.2. Initial Token

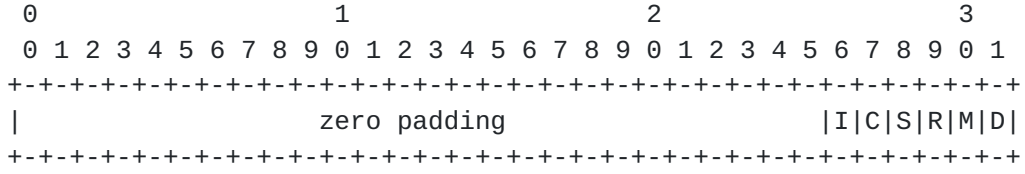
This is the initial token sent by the context initiator, generated by GSS_Init_sec_context().




:
AP-REQ data

initial token ID (32 bits)
Contains the integer 0x01010101, which identifies this as the initial token in the context setup.

flags (32 bits)
Context establishment flags, with values as defined by RFC 1509.



I	GSS_C_INTEG_FLAG	0x00000020
C	GSS_C_CONF_FLAG	0x00000010
S	GSS_C_SEQUENCE_FLAG	0x00000008
R	GSS_C_REPLAY_FLAG	0x00000004
M	GSS_C_MUTUAL_FLAG	0x00000002
D	GSS_C_DELEG_FLAG	0x00000001

The zero padding reserves bits for future expansion. The padding bits shall be set to zero by the initator and ignored by the acceptor. GSS_C_DELEG_FLAG ("D" flag) must be set if the "delegated credentials" option is included.

checksum type count (32 bits)
The number of checksum types supported by the initiator.

checksum type list (variable length)
A list of Kerberos checksum types, as defined in RFC 1510 section 6.4. These checksum types must be collision-proof and keyed with the context key. Each checksum type number shall be 32 bits wide. This list should contain all the checksum types supported by the initiator.

options (variable length)
The option format will be described later.

AP-REQ length (32 bits)
The length of the following KRB_AP_REQ message.

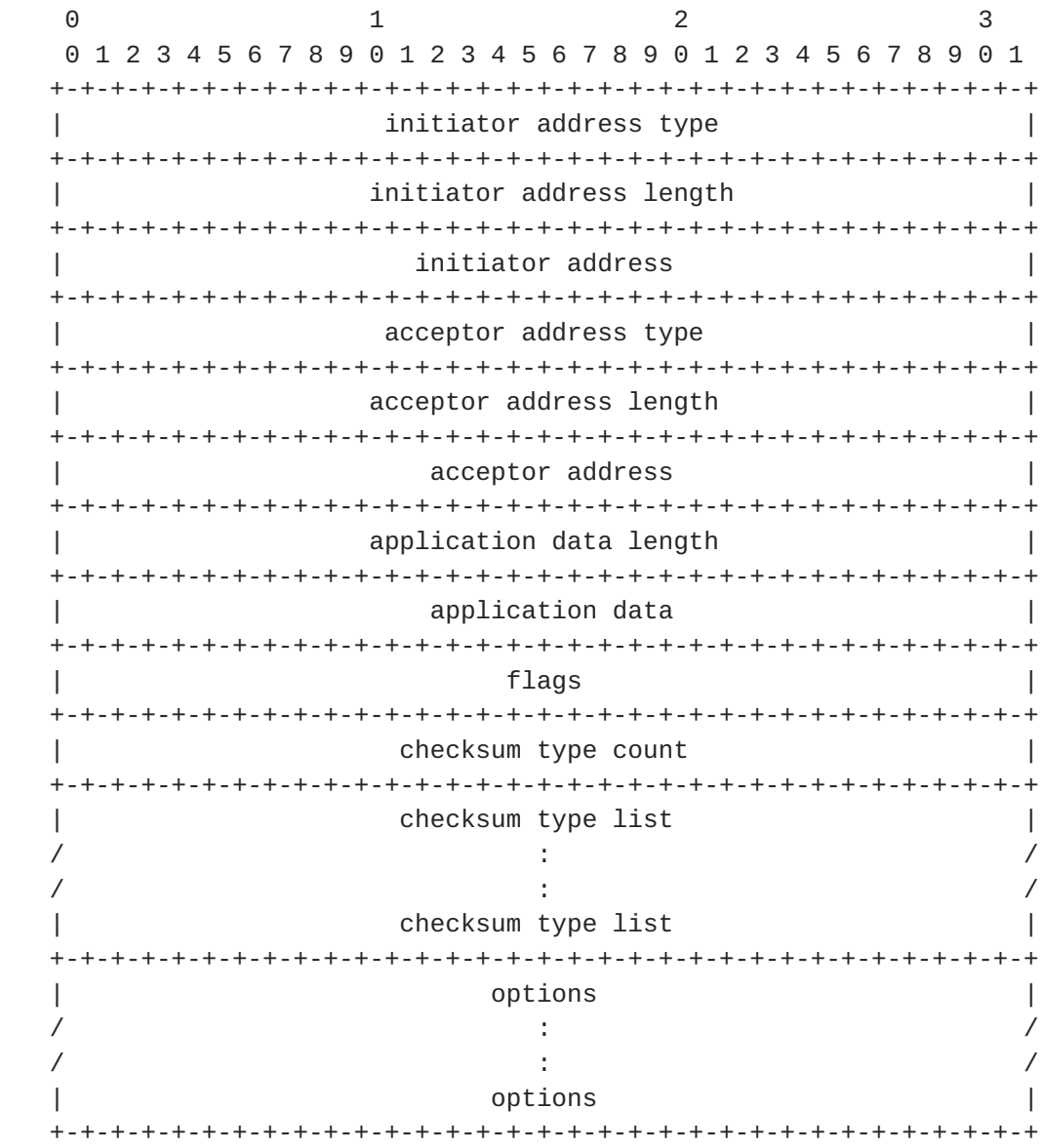
AP-REQ data (variable length)
The AP-REQ message as described in RFC 1510. The checksum in the authenticator will be computed over the items listed in the next section.

The optional sequence number field shall be used in the AP-REQ. The initiator should generate a subkey in the authenticator, and the acceptor should generate a subkey in the AP-REP. The key used for the

per-message tokens will be the AP-REP subkey, or if that is not present, the authenticator subkey, or if that is not present, the session key. When subkeys are generated, it is strongly recommended that they be of the same type as the associated session key.

2.3.2.1. Data to be Checksummed in AP-REQ

The checksum in the AP-REQ message is calculated over the following items. Like in the actual tokens, no padding should be added to force integer fields to align on 32 bit boundaries. This particular set of data should not be sent as a part of any token; it merely specifies what is to be checksummed in the AP-REQ.



initiator address type (32 bits)
The initiator address type, as defined in the Kerberos protocol specification.

initiator address length (32 bits)

The length in bytes of the following initiator address.

initiator address (variable length)
The actual initiator address, in network byte order.

acceptor address type (32 bits)
The acceptor address type, as defined in the Kerberos protocol specification.

initiator address length (32 bits)
The length in bytes of the following acceptor address.

initiator address (variable length)
The actual acceptor address, in network byte order.

application data length (32 bits)
The length of the following application data, if provided. If no application data are passed as input channel bindings, this length field should be set to zero.

applicatation data (variable length)
The actual application data, if provided.

flags (32 bits)
The context establishment flags from the initial token.

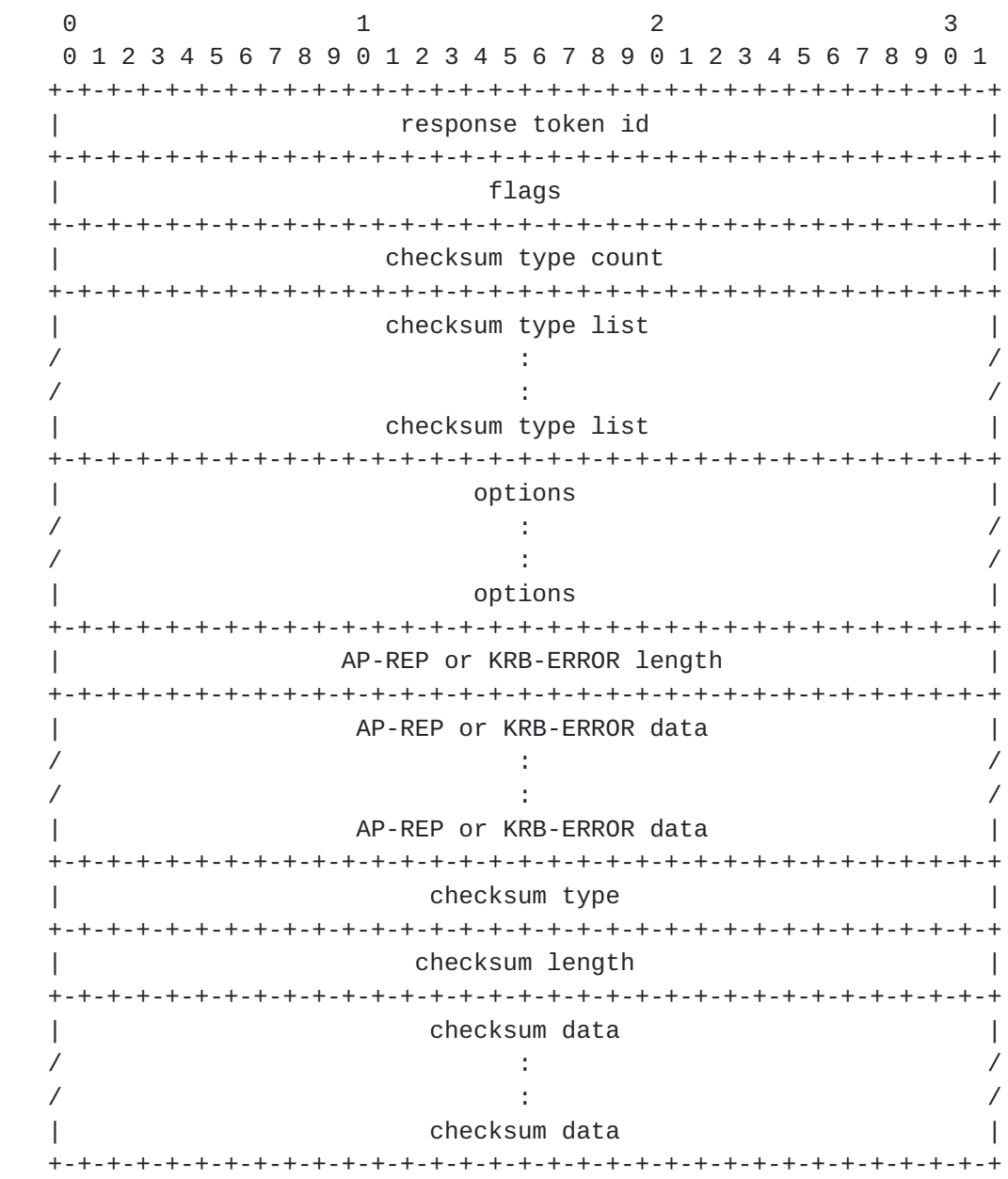
checksum type count (32 bits)
The number of checksum types supported by the initiator.

checksum type list (variable length)
The same list of checksum types contained in the initial token.

options (variable length)
The options list from the initial token.

2.3.3. Response Token

This is the reponse token sent by the context acceptor, if mutual authentication is enabled.



response token id (32 bits)

Contains the integer 0x02020202, which identifies this as the response token in the context setup.

flags (32 bits)

This currently is defined to only contain one flag, the error flag, which has a value of 0x00000001. If this flag is set, then the last field contains a KRB-ERROR message, otherwise, it contains an AP-REP message. The other bits in the flags field are reserved; they should be set to zero by the acceptor and ignored by the initiator.

checksum type count (32 bits)
     The number of checksum types supported by both the initiator and
     the acceptor.

checksum type list (variable length)
A list of Kerberos checksum types, as defined in RFC 1510 section 6.4. These checksum types must be collision-proof and keyed with the context key. Each checksum type number shall be 32 bits wide. This list should contain the intersection of the list of checksum types specified by the initiator in the initial token and the list of checksum types supported by the acceptor.

options (variable length)
The option list, as described earlier. At this time, no options are defined, but an implementation might make use of these options to acknowledge an option from the initial token. After all the options are specified, a null option must be used to terminate the list.

AP-REP or KRB-ERROR length (32 bits)
Depending on the value of the error flag, length in bytes of the AP-REP or KRB-ERROR message.

AP-REP or KRB-ERROR data (variable length)
Depending on the value of the error flag, the AP-REP or KRB-ERROR message as described in RFC 1510. If this field contains an AP-REP message, the sequence number field in the AP-REP shall be filled. If this is a KRB-ERROR message, no further fields will be in this message.

checksum type (32 bits)
A Kerberos checksum type, as defined in RFC 1510 section 6.4. This checksum type must be collision-proof and keyed with the context key. The checksum type implies the length of the checksum data.

checksum length (32 bits)
The number of bytes in the following checksum data field.

checksum data (variable length)
The checksum itself, as defined in RFC 1510 section 6.4, computed over the concatenation of the flags and the options. XXX does this need a new key usage as well?

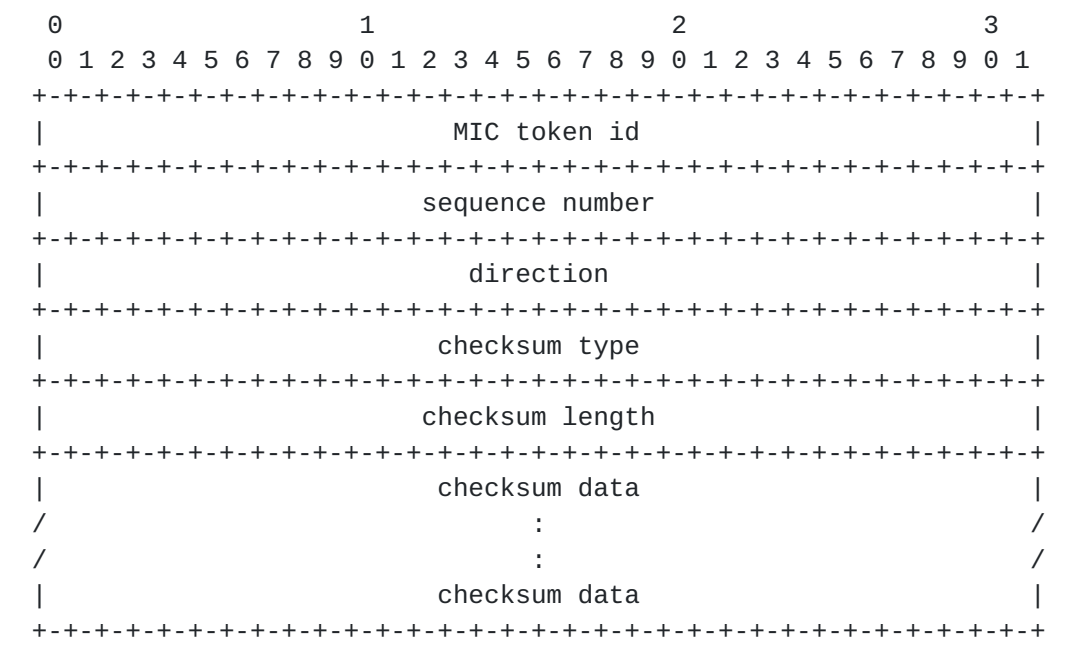
2.4. Per-message Tokens

2.4.1. Sequence Number Usage Sequence numbers for per-message tokens are 32 bit unsigned integers, which are incremented by 1 after each token. An overflow condition should result in a wraparound of the sequence number to zero. The initiator and acceptor each keep their own sequence numbers per connection.

2.4.2. MIC Token

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 when it is received. The MIC token has the following format:



MIC token id (32 bits)
Contains the integer 0x03030303, which identifies this as a MIC token.

sequence number (32 bits)
The sequence number.

direction (32 bits)
All bits in this field shall be zero if the message is sent from the context initiator. If the message is sent from the context acceptor, all bits shall be one.

checksum type (32 bits)
A Kerberos checksum type, as defined in RFC 1510 section 6.4. This checksum type must be collision-proof and keyed with the context key.

checksum length (32 bits)
The number of bytes in the following checksum data field.

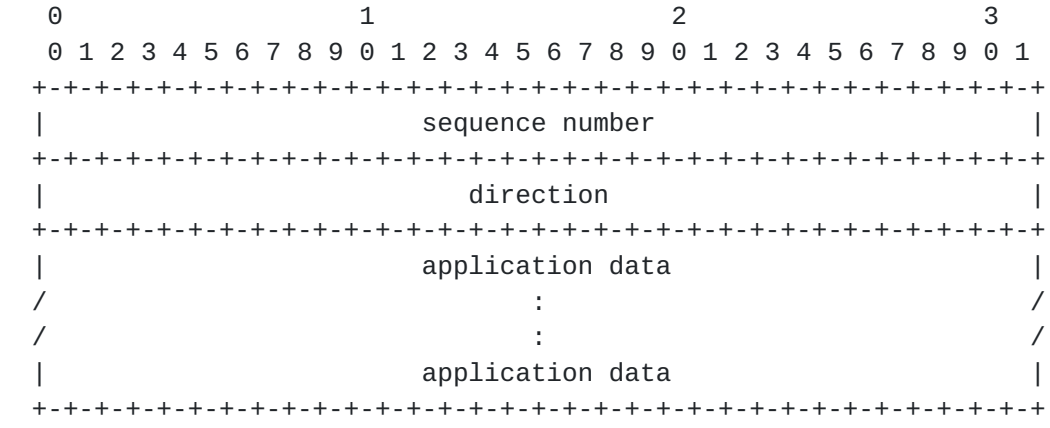
checksum data (variable length)
The checksum itself, as defined in RFC 1510 section 6.4. The checksum is calculated as in the following section.

XXX A kerberos key usage needs to be registered for generating MIC tokens.

The mechanism implementation should only use checksum types which it knows to be valid for both peers. If mutual authentication is used,

then any checksum type specified by the acceptor may be used. If mutual authentication is not used XXX what do we do then? This seems to be a more general issue than just GSSAPI. What checksum type did we use for the authenticator checksum?

2.4.2.1. Data to be Checksummed in MIC Token The checksum in the MIC token shall be calculated over the following elements.



sequence number (32 bits)
The sequence number.

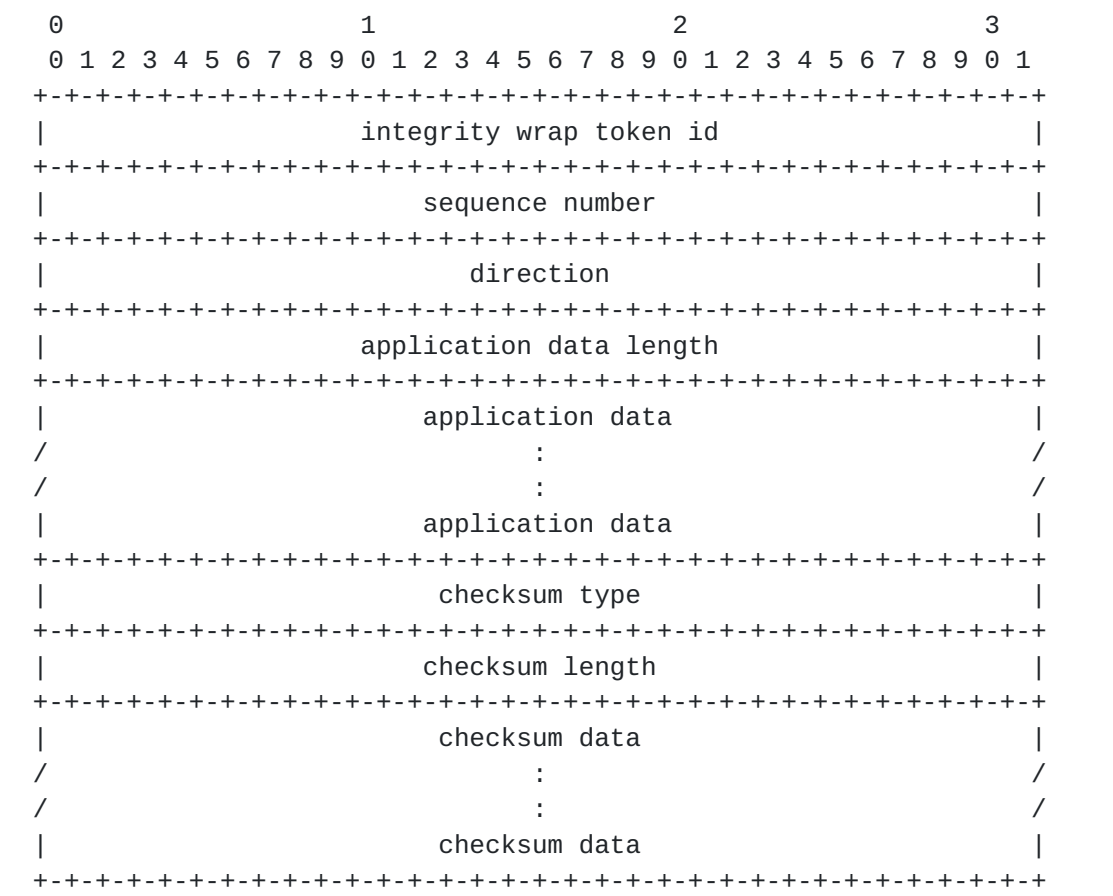
direction (32 bits)
All bits in this field shall be zero if the message is sent from the context initiator. If the message is sent from the context acceptor, all bits shall be one.

application data (variable length)
The application-supplied data.

2.4.3. Wrap Token

Use of the GSS_Wrap() call yields a token which encapsulates the input user data (optionally encrypted) along with associated integrity check quantities.

2.4.3.1. Wrap Token With Integrity Only



integrity wrap token id (32 bits)
: Contains the integer 0x04040404, which identifies this as a Wrap token with integrity only.

sequence number (32 bits)
: The sequence number.

direction (32 bits)
: All bits in this field shall be zero if the message is sent from the context initiator. If the message is sent from the context acceptor, all bits shall be one.

application data length (32 bits)
: The number of bytes in the following application data field.

application data (variable length)
: The application-supplied data.

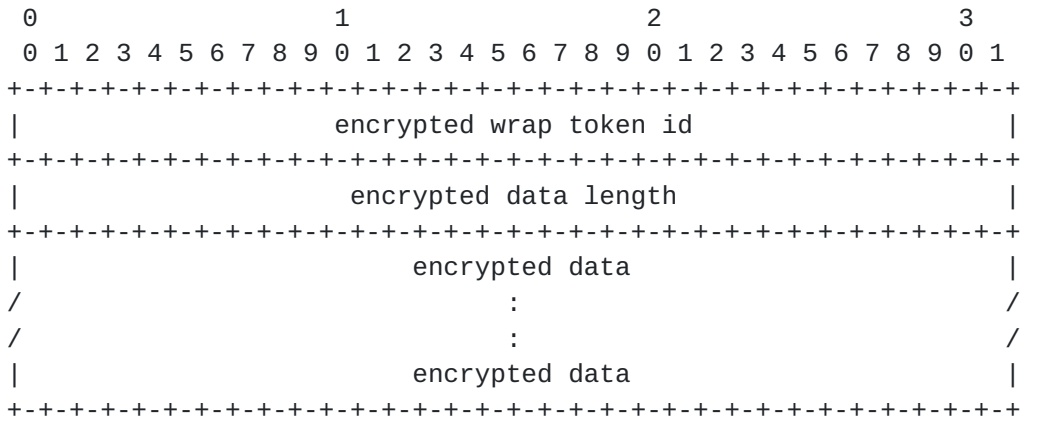
checksum type (32 bits)
: A Kerberos checksum type, as defined in RFC 1510 section 6.4. This checksum type must be collision-proof and keyed with the context key. The checksum type implies the length of the checksum data.

checksum length (32 bits)
     The number of bytes in the following checksum data field.

checksum data (variable length)
The checksum itself, as defined in RFC 1510 section 6.4, computed over the concatenation of the sequence number, direction field, and application data, as in the MIC token checksum in the previous section.

The mechanism implementation should only use checksum types which it knows to be valid for both peers, as described for MIC tokens.

2.4.3.2. Wrap Token With Integrity and Encryption

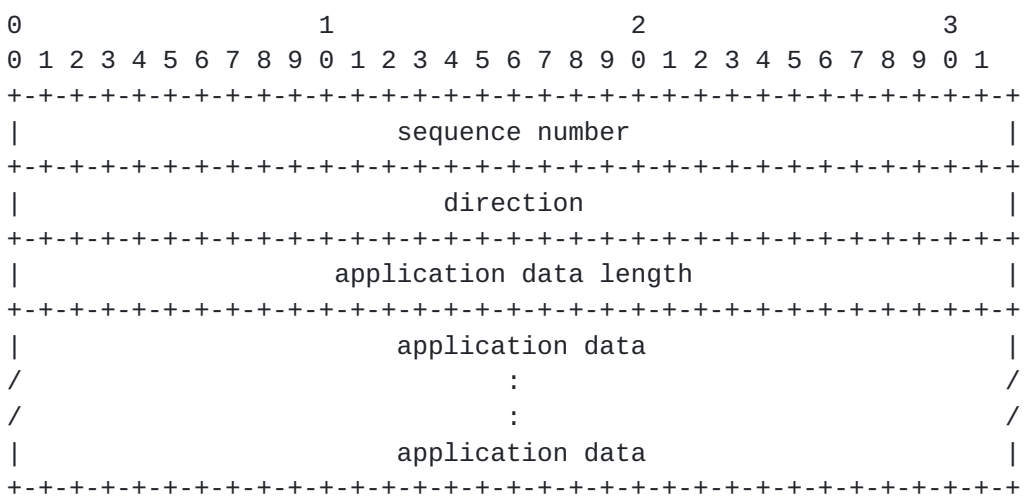


encrypted wrap token id (32 bits)
Contains the integer 0x05050505, which identifies this as a Wrap token with integrity and encryption.

encrypted data length (32 bits)
The number of bytes in the following encrypted data field.

encrypted data (variable length)
The encrypted data itself, as defined in RFC 1510 section 6.3. The encryption is performed using the key/enctype exchanged during context setup. The actual data to be encrypted are specified below.

2.4.3.2.1. Data to be Encrypted in Wrap Token



sequence number (32 bits)
The sequence number.

direction (32 bits)
All bits in this field shall be zero if the message is sent from the context initiator. If the message is sent from the context acceptor, all bits shall be one.

application data length (32 bits)
The number of bytes in the following application data field.

application data (variable length)
The application-supplied data.

XXX Two kerberos key usages need to be registered, one for integrity-only tokens, and one for integrity-and-encryption tokens.

3. Kerberos Protocol Dependencies This protocol makes several assumptions about the Kerberos protocol, which may require changes to the successor of RFC 1510.

Sequence numbers, checksum types, and address types are assumed to be no wider than 32 bits. The Kerberos protocol specification might need to be modified to accomodate this. This obviously requires some further discussion.

Key usages need to be registered within the Kerberos protocol for use with GSSAPI per-message tokens. One may also need to be registered for the checksum in the reply token during context establishment.

This protocol also makes the assumption that any cryptosystem used with the session key will include integrity protection, i.e., it assumes that no "raw" cryptosystems will be used.

4. Security Considerations The GSSAPI is a security protocol; therefore, security considerations are discussed throughout this document. The old Kerberos 5 GSSAPI mechanism's constraints on possible cryptosystems and checksum types do not permit it to be readily expanded

to accomodate more secure technologies with larger checksums or encryption block sizes. Sites are strongly encouraged to adopt the mechanism specified in this document in the light of recent publicity about the deficiencies of DES.

5. References

[RFC1510] Kohl, J., Neumann, C., "The Kerberos Network Authentication Service (V5)", RFC 1510.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964.

[RFC2078] Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078

6. Author's Address

Tom Yu
Massachusetts Institute of Technology
Room E40-345
77 Massachusetts Avenue
Cambridge, MA 02139
USA

email: tlyu@mit.edu
phone: +1 617 253 1753