INTERNET-DRAFT
draft-ietf-krb-wg-rfc1510ter-02.txt
Expires: 26 April 2006

Tom Yu
MIT

23 October 2005

The Kerberos Network Authentication Service (Version 5)

Status of This Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at

http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at

http://www.ietf.org/shadow.html

Copyright Notice

Copyright (C) The Internet Society (2005). All Rights Reserved.

Abstract

This document describes version 5 of the Kerberos network authentication protocol. It describes a framework to allow for extensions to be made to the protocol without creating interoperability problems.

Key Words for Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in RFC 2119.

Table of Contents

1. Introduction

The Kerberos network authentication protocol is a trusted-third-party protocol utilizing symmetric-key cryptography. It assumes that all communications between parties in the protocol may be arbitrarily tampered with or monitored, and that the security of the overall system depends only on the effectiveness of the cryptographic techniques and the secrecy of the cryptographic keys used. The Kerberos protocol authenticates an application client's identity to an application server, and likewise authenticates the application server's identity to the application client. These assurances are made possible by the client and the server sharing secrets with the trusted third party: the Kerberos server, also known as the Key Distribution Center (KDC). In addition, the protocol establishes an ephemeral shared secret (the session key) between the client and the server, allowing the protection of further communications between them.

The Kerberos protocol, as originally specified, provides insufficient means for extending the protocol in a backwards-compatible way. This deficiency has caused problems for interoperability. This document describes a framework which enables backwards-compatible extensions to the Kerberos protocol.

1.1. Kerberos Protocol Overview

Kerberos comprises three main sub-protocols: credentials acquisition, session key exchange, and session key usage. A client acquires credentials by asking the KDC for a credential for a service; the KDC issues the credential, which contains a ticket and a session key. The ticket, containing the client's identity, timestamps, expiration time, and a session key, is a encrypted in a key known to the application server. The KDC encrypts the credential using a key known to the client, and transmits the credential to the client.

There are two means of requesting credentials: the Authentication Service (AS) exchange, and the Ticket-Granting Service (TGS) exchange. In the typical AS exchange, a client uses a password-derived key to decrypt the response. In the TGS exchange, the KDC behaves as an application server; the client authenticates to the TGS by using a Ticket-Granting Ticket (TGT). The client usually obtains the TGT by using the AS exchange.

Session key exchange consists of the client transmitting the ticket to the application server, accompanied by an authenticator. The authenticator contains a timestamp and additional data encrypted using the ticket's session key. The application server decrypts the ticket, extracting the session key. The application server then uses the session key to decrypt the authenticator. Upon successful

decryption of the authenticator, the application server knows that the data in the authenticator were sent by the client named in the

associated ticket. Additionally, since authenticators expire more quickly than tickets, the application server has some assurance that the transaction is not a replay. The application server may send an encrypted acknowledgment to the client, verifying its identity to the client.

Once session key exchange has occurred, the client and server may use the established session key to protect further traffic. This protection may consist of protection of integrity only, or of protection of confidentiality and integrity. Additional measures exist for a client to securely forward credentials to a server.

The entire scheme depends on loosely synchronized clocks. Synchronization of the clock on the KDC with the application server clock allows the application server to accurately determine whether a credential is expired. Likewise, synchronization of the clock on the client with the application server clock prevents replay attacks utilizing the same credential. Careful design of the application protocol may allow replay prevention without requiring client-server clock synchronization.

After establishing a session key, application client and the application server can exchange Kerberos protocol messages that use the session key to protect the integrity or confidentiality of communications between the client and the server. Additionally, the client may forward credentials to the application server.

The credentials acquisition protocol takes place over specific, defined transports (UDP and TCP). Application protocols define which transport to use for the session key establishment protocol and for messages using the session key; the application may choose to perform its own encapsulation of the Kerberos messages, for example.

1.2. Document Organization

The remainder of this document begins by describing the general frameworks for protocol extensibility, including whether to interpret unknown extensions as critical. It then defines the protocol messages and exchanges.

The definition of the Kerberos protocol uses Abstract Syntax Notation One (ASN.1) [X680], which specifies notation for describing the abstract content of protocol messages. This document defines a number of base types using ASN.1; these base types subsequently appear in multiple types which define actual protocol messages. Definitions of principal names and of tickets, which are central to the protocol, also appear preceding the protocol message definitions.

2. Compatibility Considerations

2.1. Extensibility

In the past, significant interoperability problems have resulted from conflicting assumptions about how the Kerberos protocol can be extended. As the deployed base of Kerberos grows, the ability to extend the Kerberos protocol becomes more important. In order to ensure that vendors and the IETF can extend the protocol while maintaining backwards compatibility, this document outlines a framework for extending Kerberos.

Kerberos provides two general mechanisms for protocol extensibility. Many protocol messages, including some defined in RFC 1510, contain typed holes--sub-messages containing an octet string along with an integer that identifies how to interpret the octet string. The integer identifiers are registered centrally, but can be used both for vendor extensions and for extensions standardized in the IETF. This document adds typed holes to a number of messages which previously lacked typed holes.

Many new messages defined in this document also contain ASN.1 extension markers. These markers allow future revisions of this document to add additional elements to messages, for cases where typed holes are inadequate for some reason. Because tag numbers and position in a sequence need to be coordinated in order to maintain interoperability, implementations MUST NOT include ASN.1 extensions except when those extensions are specified by IETF standards-track documents.

2.2. Compatibility with RFC 1510

Implementations of RFC 1510 did not use extensible ASN.1 types. Sending additional fields not in RFC 1510 to these implementations results in undefined behavior. Examples of this behavior are known to include discarding messages with no error indications.

Where messages have been changed since RFC 1510, ASN.1 CHOICE types are used; one alternative of the CHOICE provides a message which is wire-encoding compatible with RFC 1510, and the other alternative provides the new, extensible message.

Implementations sending new messages MUST ensure that the recipient supports these new messages. Along with each extensible message is a guideline for when that message MAY be used. If that guideline is followed, then the recipient is guaranteed to understand the message.

2.3. Backwards Compatibility

This document describes two sets (for the most part) of ASN.1 types. The first set of types is wire-encoding compatible with RFC 1510 and [KCLAR]. The second set of types is the set of types enabling

extensibility. This second set may be referred to as "extensibility-

enabled types". [need to make this consistent throughout?]

A major difference between the new extensibility-enabled types and the types for RFC 1510 compatibility is that the extensibility-enabled types allow for the use of UTF-8 encodings in various character strings in the protocol. Each party in the protocol must have some knowledge of the capabilities of the other parties in the protocol. There are methods for establishing this knowledge without necessarily requiring explicit configuration.

An extensibility-enabled client can detect whether a KDC supports the extensibility-enabled types by requesting an extensibility-enabled reply. If the KDC replies with an extensibility-enabled reply, the client knows that the KDC supports extensibility. If the KDC issues an extensibility-enabled ticket, the client knows that the service named in the ticket is extensibility-enabled.

2.4. Sending Extensible Messages

Care must be taken to make sure that old implementations can understand messages sent to them even if they do not understand an extension that is used. Unless the sender knows the extension is supported, the extension cannot change the semantics of the core message or previously defined extensions.

For example, an extension including key information necessary to decrypt the encrypted part of a KDC-REP could only be used in situations where the recipient was known to support the extension. Thus when designing such extensions it is important to provide a way for the recipient to notify the sender of support for the extension. For example in the case of an extension that changes the KDC-REP reply key, the client could indicate support for the extension by including a padata element in the AS-REQ sequence. The KDC should only use the extension if this padata element is present in the AS-REQ. Even if policy requires the use of the extension, it is better to return an error indicating that the extension is required than to use the extension when the recipient may not support it; debugging why implementations do not interoperate is easier when errors are returned.

2.5. Criticality

Recipients of unknown message extensions (including typed holes, new flags, and ASN.1 extension elements) should preserve the encoding of the extension but otherwise ignore the presence of the extension; i.e., unknown extensions SHOULD be treated as non-critical. If a copy of the message is used later--for example, when a Ticket received in a KDC-REP is later used in an AP-REQ--then the unknown extensions MUST be included.

An implementation SHOULD NOT reject a request merely because it does not understand some element of the request. As a related consequence, implementations SHOULD handle communicating with other implementations which do not implement some requested options. This may require designers of options to provide means to determine whether an option has been rejected, not understood, or (perhaps maliciously) deleted or modified in transit.

There is one exception to non-criticality described above: if an unknown authorization data element is received by a server either in an AP-REQ or in a Ticket contained in an AP-REQ, then the authentication SHOULD fail. Authorization data is intended to restrict the use of a ticket. If the service cannot determine whether the restriction applies to that service then a security weakness may result if authentication succeeds. Authorization elements meant to be truly optional can be enclosed in the AD-IF-RELEVANT element.

Many RFC 1510 implementations ignore unknown authorization data elements. Depending on these implementations to honor authorization data restrictions may create a security weakness.

2.6. Authenticating Cleartext Portions of Messages

Various denial of service attacks and downgrade attacks against Kerberos are possible unless plaintexts are somehow protected against modification. An early design goal of Kerberos Version 5 was to avoid encrypting more of the authentication exchange that was required. (Version 4 doubly-encrypted the encrypted part of a ticket in a KDC reply, for example.) This minimization of encryption reduces the load on the KDC and busy servers. Also, during the initial design of Version 5, the existence of legal restrictions on the export of cryptography made it desirable to minimize of the number of uses of encryption in the protocol. Unfortunately, performing this minimization created numerous instances of unauthenticated security-relevant plaintext fields.

The extensible variants of the messages described in this document wrap the actual message in an ASN.1 sequence containing a keyed checksum of the contents of the message. Guidelines in [XXX] section 3 specify when the checksum MUST be included and what key MUST be used. Guidelines on when to include a checksum are never ambiguous: a particular PDU is never correct both with and without a checksum. With the exception of the KRB-ERROR message, receiving implementations MUST reject messages where a checksum is included and not expected or where a checksum is expected but not included. The receiving implementation does not always have sufficient information to know whether a KRB-ERROR should contain a checksum. Even so, KRB-ERROR messages with invalid checksums MUST be rejected and

implementations MAY consider the presence or absence of a checksum when evaluating whether to trust the error.

This authenticated cleartext protection is provided only in the extensible variants of the messages; it is never used when communicating with an RFC 1510 implementation.

2.7. Capability Negotiation

Kerberos is a three-party protocol. Each of the three parties involved needs a means of detecting the capabilities supported by the others. Two of the parties, the KDC and the application server, do not communicate directly in the Kerberos protocol. Communicating capabilities from the KDC to the application server requires using a ticket as an intermediary.

The main capability requiring negotiation is the support of the extensibility framework described in this document. Negotiation of this capability while remaining compatible with RFC 1510 implementations is possible. The main complication is that the client needs to know whether the application server supports the extensibility framework prior to sending any message to the application server. This can be accomplished if the KDC has knowledge of whether an application server supports the extensibility framework.

Client software advertizes its capabilities when requesting credentials from the KDC. If the KDC recognizes the capabilities, it acknowledges this fact to the client in its reply. In addition, if the KDC has knowledge that the application server supports certain capabilities, it also communicates this knowledge to the client in its reply. The KDC can encode its own capabilities in the ticket so that the application server may discover these capabilities. The client advertizes its capabilities to the application server when it initiates authentication to the application server.

3. Use of ASN.1 in Kerberos

Kerberos uses the ASN.1 Distinguished Encoding Rules (DER) [X690]. Even though ASN.1 theoretically allows the description of protocol messages to be independent of the encoding rules used to encode the messages, Kerberos messages MUST be encoded with DER. Subtleties in the semantics of the notation, such as whether tags carry any semantic content to the application, may cause the use of other ASN.1 encoding rules to be problematic.

Implementors not using existing ASN.1 tools (e.g., compilers or support libraries) are cautioned to thoroughly read and understand the actual ASN.1 specification to ensure correct implementation behavior. There is more complexity in the notation than is immediately obvious, and some tutorials and guides to ASN.1 are misleading or erroneous. Recommended tutorials and guides include

[Dub00], [Lar99], though there is still no substitute for reading the actual ASN.1 specification.

3.1. Module Header

The type definitions in this document assume an ASN.1 module definition of the following form:

```
KerberosV5Spec3 {
    iso(1) identified-organization(3) dod(6) internet(1)
    security(5) kerberosV5(2) modules(4) krb5spec3(4)
} DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- Rest of definitions here

END
```

This specifies that the tagging context for the module will be explicit and that automatic tagging is not done.

Some other publications [RFC1510] [RFC1964] erroneously specify an object identifier (OID) having an incorrect value of "5" for the "dod" component of the OID. In the case of RFC 1964, use of the "correct" OID value would result in a change in the wire protocol; therefore, the RFC 1964 OID remains unchanged for now.

3.2. Top-Level Type

The ASN.1 type "KRB-PDU" is a CHOICE over all the types (Protocol Data Units or PDUs) which an application may directly reference. Applications SHOULD NOT transmit any types other than those which are alternatives of the KRB-PDU CHOICE.

```
-- top-level type
--
-- Applications should not directly reference any types
-- other than KRB-PDU and its component types.
--
KRB-PDU         ::= CHOICE {
    ticket      Ticket,
    as-req      AS-REQ,
    as-rep      AS-REP,
    tgs-req     TGS-REQ,
    tgs-rep     TGS-REP,
    ap-req      AP-REQ,
    ap-rep      AP-REP,
    krb-safe    KRB-SAFE,
    krb-priv    KRB-PRIV,
    krb-cred    KRB-CRED,
    tgt-req     TGT-REQ,
    krb-error   KRB-ERROR,
    ...
}
```

3.3. Previously Unused ASN.1 Notation (informative)

Some aspects of ASN.1 notation used in this document were not used in [KCLAR], and may be unfamiliar to some readers. This subsection is informative; for normative definitions of the notation, see the actual ASN.1 specifications [X680], [X682], [X683].

3.3.1. Parameterized Types

This document uses ASN.1 parameterized types [X683] to make definitions of types more readable. For some types, some or all of the parameters are advisory, i.e., they are not encoded in any form for transmission in a protocol message. These advisory parameters can describe implementation behavior associated with the type.

3.3.2. Constraints

This document uses ASN.1 constraints, including the "UserDefinedConstraint" notation [X682]. Some constraints can be handled automatically by tools that can parse them. Uses of the "UserDefinedConstraint" notation (the "CONSTRAINED BY" notation) will typically need to have behavior manually coded; the notation provides a formalized way of conveying intended implementation behavior.

The "WITH COMPONENTS" constraint notation allows constraints to apply to component types of a SEQUENCE type. This constraint notation effectively allows constraints to "reach into" a type to constrain its component types.

3.4. New Types

This document defines a number of ASN.1 types which are new since [KCLAR]. The names of these types will typically have a suffix like "Ext", indicating that the types are intended to support extensibility. Types original to RFC 1510 and [KCLAR] have been renamed to have a suffix like "1510". The "Ext" and "1510" types often contain a number of common elements, but differ primarily in the way strings are encoded.

4. Basic Types

These "basic" Kerberos ASN.1 types appear in multiple other Kerberos types.

4.1. Constrained Integer Types

In RFC 1510, many types contained references to the unconstrained INTEGER type. Since an unconstrained INTEGER can contain almost any possible abstract integer value, most of the unconstrained references to INTEGER in RFC 1510 were constrained to 32 bits or smaller in

[KCLAR].

```
   -- signed values representable in 32 bits
   --
   -- These are often used as assigned numbers for various things.
   Int32           ::= INTEGER (-2147483648..2147483647)
```

The "Int32" type often contains an assigned number identifying the contents of a typed hole. Unless otherwise stated, non-negative values are registered, and negative values are available for local use.

```
   -- unsigned 32 bit values
   UInt32          ::= INTEGER (0..4294967295)
```

The "UInt32" type is used in some places where an unsigned 32-bit integer is needed.

```
   -- unsigned 64 bit values
   UInt64          ::= INTEGER (0..18446744073709551615)
```

The "UInt64" type is used in places where 32 bits of precision may provide inadequate security.

```
   -- sequence numbers
   SeqNum          ::= UInt64
```

Sequence numbers were constrained to 32 bits in [KCLAR], but this document allows for 64-bit sequence numbers.

```
   -- nonces
   Nonce           ::= UInt64
```

Likewise, nonces were constrained to 32 bits in [KCLAR], but expanded to 64 bits here.

```
   -- microseconds
   Microseconds    ::= INTEGER (0..999999)
```

The "microseconds" type is intended to carry the microseconds part of a time value.

4.2. KerberosTime

```
   KerberosTime    ::= GeneralizedTime (CONSTRAINED BY {
                           -- MUST NOT include fractional seconds
   })
```

The timestamps used in Kerberos are encoded as GeneralizedTimes. A KerberosTime value MUST NOT include any fractional portions of the seconds. As required by the DER, it further MUST NOT include any separators, and it specifies the UTC time zone (Z). Example: The

only valid format for UTC time 6 minutes, 27 seconds after 9 pm on 6

November 1985 is "19851106210627Z".

4.3. KerberosString

```
-- used for names and for error messages
KerberosString  ::= CHOICE {
    ia5         GeneralString (IA5String),
    utf8        UTF8String,
    ...         -- no extension may be sent
                -- to an rfc1510 implementation --
}
```

The KerberosString type is used for character strings in various places in the Kerberos protocol. For compatibility with RFC 1510, GeneralString values constrained to IA5String (US-ASCII) are permitted in messages exchanged with RFC 1510 implementations. The new protocol messages contain strings encoded as UTF-8, and these strings MUST be normalized using [SASLPREP]. KerberosString values are present in principal names and in error messages. Control characters SHOULD NOT be used in principal names, and used with caution in error messages.

```
-- IA5 choice only; useful for constraints
KerberosStringIA5       ::= KerberosString
    (WITH COMPONENTS { ia5 PRESENT })

-- IA5 excluded; useful for constraints
KerberosStringExt       ::= KerberosString
    (WITH COMPONENTS { ia5 ABSENT })
```

KerberosStringIA5 requires the use of the "ia5" alternative, while KerberosStringExt forbids it. These types appear in ASN.1 constraints on messages.

For detailed background regarding the history of character string use in Kerberos, as well as discussion of some compatibility issues, see Appendix B.

4.4. Language Tags

```
-- used for language tags
LangTag ::= PrintableString
    (FROM ("A".."Z" | "a".."z" | "0".."9" | "-"))
```

The "LangTag" type is used to specify language tags for localization purposes, using the [RFC3066] format.

4.5. KerberosFlags

For several message types, a specific constrained bit string type,

KerberosFlags, is used.

```
KerberosFlags { NamedBits } ::= BIT STRING (SIZE (32..MAX))
    (CONSTRAINED BY {
    -- MUST be a valid value of -- NamedBits
    -- but if the value to be sent would be truncated to shorter
    -- than 32 bits according to DER, the value MUST be padded
    -- with trailing zero bits to 32 bits.  Otherwise, no
    -- trailing zero bits may be present.

})
```

The actual bit string type encoded in Kerberos messages does not use named bits. The advisory parameter of the KerberosFlags type names a bit string type defined using named bits, whose value is encoded as if it were a bit string with unnamed bits. This practice is necessary because the DER require trailing zero bits to be removed when encoding bit strings defined using named bits. Existing implementations of Kerberos send exactly 32 bits rather than truncating, so the size constraint requires the transmission of at least 32 bits. Trailing zero bits beyond the first 32 bits are truncated.

4.6. Typed Holes

```
-- Typed hole identifiers
TH-id           ::= CHOICE {
    int32               Int32,
    rel-oid             RELATIVE-OID
}
```

The "TH-id" type is a generalized means to identify the contents of a typed hole. The "int32" alternative may be used for simple integer assignments, in the same manner as "Int32", while the "rel-oid" alternative may be used for hierarchical delegation.

4.7. HostAddress and HostAddresses

```
AddrType        ::= Int32

HostAddress     ::= SEQUENCE  {
    addr-type   [0] AddrType,
    address     [1] OCTET STRING
}

-- NOTE: HostAddresses is always used as an OPTIONAL field and
-- should not be a zero-length SEQUENCE OF.
--
-- The extensible messages explicitly constrain this to be
-- non-empty.
HostAddresses   ::= SEQUENCE OF HostAddress
```

addr-type
This field specifies the type of address that follows.

address
This field encodes a single address of the type identified by "addr-type".

All negative values for the host address type are reserved for local use. All non-negative values are reserved for officially assigned type fields and interpretations.

addr-type	meaning
2	IPv4
3	directional
12	DECnet
20	NetBIOS
24	IPv6

4.7.1. Internet (IPv4) Addresses

Internet (IPv4) addresses are 32-bit (4-octet) quantities, encoded in MSB order (most significant byte first). The IPv4 loopback address SHOULD NOT appear in a Kerberos PDU. The type of IPv4 addresses is two (2).

4.7.2. Internet (IPv6) Addresses

IPv6 addresses [RFC2373] are 128-bit (16-octet) quantities, encoded in MSB order (most significant byte first). The type of IPv6 addresses is twenty-four (24). The following addresses MUST NOT appear in any Kerberos PDU:

* the Unspecified Address

* the Loopback Address

* Link-Local addresses

This restriction applies to the inclusion in the address fields of Kerberos PDUs, but not to the address fields of packets that might carry such PDUs. The restriction is necessary because the use of an address with non-global scope could allow the acceptance of a message sent from a node that may have the same address, but which is not the host intended by the entity that added the restriction. If the link-local address type needs to be used for communication, then the address restriction in tickets must not be used (i.e. addressless

tickets must be used).

IPv4-mapped IPv6 addresses MUST be represented as addresses of type 2.

4.7.3. DECnet Phase IV addresses

DECnet Phase IV addresses are 16-bit addresses, encoded in LSB order. The type of DECnet Phase IV addresses is twelve (12).

4.7.4. Netbios addresses

Netbios addresses are 16-octet addresses typically composed of 1 to 15 alphanumeric characters and padded with the US-ASCII SPC character (code 32). The 16th octet MUST be the US-ASCII NUL character (code 0). The type of Netbios addresses is twenty (20).

4.7.5. Directional Addresses

In many environments, including the sender address in KRB-SAFE and KRB-PRIV messages is undesirable because the addresses may be changed in transport by network address translators. However, if these addresses are removed, the messages may be subject to a reflection attack in which a message is reflected back to its originator. The directional address type provides a way to avoid transport addresses and reflection attacks. Directional addresses are encoded as four byte unsigned integers in network byte order. If the message is originated by the party sending the original AP-REQ message, then an address of 0 SHOULD be used. If the message is originated by the party to whom that AP-REQ was sent, then the address 1 SHOULD be used. Applications involving multiple parties can specify the use of other addresses.

Directional addresses MUST only be used for the sender address field in the KRB-SAFE or KRB-PRIV messages. They MUST NOT be used as a ticket address or in a AP-REQ message. This address type SHOULD only be used in situations where the sending party knows that the receiving party supports the address type. This generally means that directional addresses may only be used when the application protocol requires their support. Directional addresses are type (3).

5. Principals

Principals are participants in the Kerberos protocol. A "realm" consists of principals in one administrative domain, served by one KDC (or one replicated set of KDCs). Each principal name has an arbitrary number of components, though typical principal names will only have one or two components. A principal name is meant to be readable by and meaningful to humans, especially in a realm lacking a centrally adminstered authorization infrastructure.

5.1. Name Types

Each PrincipalName has NameType indicating what sort of name it is. The name-type SHOULD be treated as a hint. Ignoring the name type, no two names can be the same (i.e., at least one of the components, or the realm, must be different).

```
-- assigned numbers for name types (used in principal names)
NameType        ::= Int32

-- Name type not known
nt-unknown              NameType ::= 0
-- Just the name of the principal as in DCE, or for users
nt-principal            NameType ::= 1
-- Service and other unique instance (krbtgt)
nt-srv-inst             NameType ::= 2
-- Service with host name as instance (telnet, rcommands)
nt-srv-hst              NameType ::= 3
-- Service with host as remaining components
nt-srv-xhst             NameType ::= 4
-- Unique ID
nt-uid                  NameType ::= 5
-- Encoded X.509 Distingished name [RFC 2253]
nt-x500-principal       NameType ::= 6
-- Name in form of SMTP email name (e.g. user@foo.com)
nt-smtp-name            NameType ::= 7
-- Enterprise name - may be mapped to principal name
nt-enterprise           NameType ::= 10
```

5.2. Principal Type Definition

The "PrincipalName" type takes a parameter to constrain which string type it contains.

```
PrincipalName { StrType }       ::= SEQUENCE {
    name-type   [0] NameType,
    -- May have zero elements, or individual elements may be
    -- zero-length, but this is NOT RECOMMENDED.
    name-string [1] SEQUENCE OF KerberosString (StrType)
}
```

The constrained types have their own names.

```
-- IA5 only
PrincipalNameIA5 ::= PrincipalName { KerberosStringIA5 }
-- IA5 excluded
PrincipalNameExt ::= PrincipalName { KerberosStringExt }
-- Either one?
```

```
PrincipalNameEither ::= PrincipalName { KerberosString }
```

name-type
hint of the type of name that follows

name-string
The "name-string" encodes a sequence of components that form a name, each component encoded as a KerberosString. Taken together, a PrincipalName and a Realm form a principal identifier. Most PrincipalNames will have only a few components (typically one or two).

5.3. Principal Name Reuse

Realm administrators SHOULD use extreme caution when considering reusing a principal name. A service administrator might explicitly enter principal names into a local access control list (ACL) for the service. If such local ACLs exist independently of a centrally administered authorization infrastructure, realm administrators SHOULD NOT reuse principal names until confirming that all extant ACL entries referencing that principal name have been updated. Failure to perform this check can result in a security vulnerability, as a new principal can inadvertently inherit unauthorized privileges upon receiving a reused principal name. An organization whose Kerberos-authenticated services all use a centrally-adminstered authorization infrastructure may not need to take these precautions regarding principal name reuse.

5.4. Best Common Practice Recommendations for the Processing of Principal Names Consisting of Internationalized Domain Names:

Kerberos principals are often created for the purpose of authenticating a service located on a machine identified by an domain name. Unfortunately, once a principal name is created it is impossible to know the source from which the resulting KerberosString was derived. It is therefore required that principal names containing internationalized domain names be processed via the following procedure:

* ensure that the IDN component must be a valid domain name as per the rules of IDNA [RFC3490]

* separate the IDN component into labels separated by any of the Full Stop characters

* fold all Full Stop characters to Full Stop (0x002E)

* for each label (perform all steps):

  o if the label begins with an ACE prefix as registered with IANA, the prefix will be removed and the rest of the label will be converted from the ACE representation to Unicode [need

reference]

- o if the label consists of one or more internationalized characters separately apply the NamePrep and then the SASLprep string preparation methods.

- o if the label consists of zero internalizationalized characters, the label is to be lower-cased

- o if the output of the two methods match, continue on to the next label; otherwise reject the principal name as invalid

* the result of a valid principal name component derived from an IDN is the joining of the individual string prepared labels separated by the Full Stop (0x002E)

5.5. Realm Names

```
Realm { StrType }       ::= KerberosString (StrType)

-- IA5 only
RealmIA5                ::= Realm { KerberosStringIA5 }

-- IA5 excluded
RealmExt                ::= Realm { KerberosStringExt }

-- Either
RealmEither             ::= Realm { KerberosString }
```

Kerberos realm names are KerberosStrings. Realms MUST NOT contain a character with the code 0 (the US-ASCII NUL). Most realms will usually consist of several components separated by periods (.), in the style of Internet Domain Names, or separated by slashes (/) in the style of X.500 names.

5.6. Best Common Practice Recommendations for the Processing of Internationalized Domain-Style Realm Names:

Domain Style Realm names are defined as all realm names whose components are separated by Full Stop (0x002E) (aka periods, '.') and contain neither colons, name containing one or more internationalized characters (not included in US-ASCII), this procedure must be used:

* the realm name must be a valid domain name as per the rules of IDNA [RFC3490]

* the following string preparation routine must be followed:

  - separate the string into components separated by any of the Full Stop characters

- fold all Full Stop characters to Full Stop (0x002E)

- for each component (perform all steps):

  o if the component begins with an ACE prefix as registered with IANA, the prefix will be removed and the rest of the component will be converted from the ACE representation to Unicode [need reference]

  o if the component consists of one or more internationalized characters separately apply the NamePrep and SASLprep string preparation methods.

    if the output of the two methods match, continue on to the next component; otherwise reject the realm name as invalid

    -

  the result of a valid realm name is the joining of the individual string prepared components separated by the Full Stop (0x002E)

In [KCLAR], the recommendation is that all domain style realm names be represented in uppercase. This recommendation is modified in the following manner. All components of domain style realm names not including internationalized characters should be upper-cased. All components of domain style realm names including internationalized characters must be lower-cased. (The lower case representation of internationalized components is enforced by the requirement that the output of NamePrep and StringPrep string preparation must be equivalent.)

5.7. Printable Representations of Principal Names

[perhaps non-normative?]

The printable form of a principal name consists of the concatenation of components of the PrincipalName value using the slash character (/), followed by an at-sign (@), followed by the realm name. Any occurrence of a backslash (\), slash (/) or at-sign (@) in the PrincipalName value is quoted by a backslash.

5.8. Ticket-Granting Service Principal

The PrincipalName value corresponding to a ticket-granting service has two components: the first component is the string "krbtgt", and the second component is the realm name of the TGS which will accept a ticket-granting ticket having this service principal name. The realm name of service always indicates which realm issued the ticket. A ticket-granting ticket issued by "A.EXAMPLE.COM" which is valid for obtaining tickets in the same realm would have the following ASN.1

values for its "realm" and "sname" components, respectively:

```
-- Example Realm and PrincipalName for a TGS

tgtRealm1       Realm ::= ia5 : "A.EXAMPLE.COM"

tgtPrinc1       PrincipalName ::= {
    name-type nt-srv-inst,
    name-string { ia5 : "krbtgt", ia5 : "A.EXAMPLE.COM" }
}
```

Its printable representation would be written as "krbtgt/A.EXAMPLE.COM@A.EXAMPLE.COM".

5.8.1. Cross-Realm TGS Principals

It is possible for a principal in one realm to authenticate to a service in another realm. A KDC can issue a cross-realm ticket-granting ticket to allow one of its principals to authenticate to a service in a foreign realm. For example, the TGS principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM" is a principal that permits a client principal in the realm A.EXAMPLE.COM to authenticate to a service in the realm B.EXAMPLE.COM. When the KDC for B.EXAMPLE.COM issues a ticket to a client originating in A.EXAMPLE.COM, the client's realm name remains "A.EXAMPLE.COM", even though the service principal will have the realm "B.EXAMPLE.COM".

6. Types Relating to Encryption

Many Kerberos protocol messages contain encrypted encodings of various data types. Some Kerberos protocol messages also contain checksums (signatures) of encodings of various types.

6.1. Assigned Numbers for Encryption

Encryption algorithm identifiers and key usages both have assigned numbers, described in [KCRYPTO].

6.1.1. EType

EType is the integer type for assigned numbers for encryption algorithms. Defined in [KCRYPTO].

```
      -- Assigned numbers denoting encryption mechanisms.
      EType ::= Int32

      -- assigned numbers for encryption schemes
      et-des-cbc-crc                  EType ::= 1
      et-des-cbc-md4                  EType ::= 2
      et-des-cbc-md5                  EType ::= 3
      --     [reserved]                        4
      et-des3-cbc-md5                 EType ::= 5
      --     [reserved]                        6
      et-des3-cbc-sha1                EType ::= 7
      et-dsaWithSHA1-CmsOID           EType ::= 9
      et-md5WithRSAEncryption-CmsOID  EType ::= 10
      et-sha1WithRSAEncryption-CmsOID EType ::= 11
      et-rc2CBC-EnvOID                EType ::= 12
      et-rsaEncryption-EnvOID         EType ::= 13
      et-rsaES-OAEP-ENV-OID           EType ::= 14
      et-des-ede3-cbc-Env-OID         EType ::= 15
      et-des3-cbc-sha1-kd             EType ::= 16
      -- AES
      et-aes128-cts-hmac-sha1-96      EType ::= 17
      -- AES
      et-aes256-cts-hmac-sha1-96      EType ::= 18
      -- Microsoft
      et-rc4-hmac                     EType ::= 23
      -- Microsoft
      et-rc4-hmac-exp                 EType ::= 24
      -- opaque; PacketCable
      et-subkey-keymaterial           EType ::= 65
```

6.1.2. Key Usages

KeyUsage is the integer type for assigned numbers for key usages. Key usage values are inputs to the encryption and decryption functions described in [KCRYPTO].

```
      -- Assigned numbers denoting key usages.
      KeyUsage ::= UInt32

      --
      -- Actual identifier names are provisional and subject to
      -- change.
      --
      ku-pa-enc-ts                    KeyUsage ::= 1
      ku-Ticket                       KeyUsage ::= 2
      ku-EncASRepPart                 KeyUsage ::= 3
      ku-TGSReqAuthData-sesskey       KeyUsage ::= 4
      ku-TGSReqAuthData-subkey        KeyUsage ::= 5
      ku-pa-TGSReq-cksum              KeyUsage ::= 6
      ku-pa-TGSReq-authenticator      KeyUsage ::= 7
      ku-EncTGSRepPart-sesskey        KeyUsage ::= 8
      ku-EncTGSRepPart-subkey         KeyUsage ::= 9
      ku-Authenticator-cksum          KeyUsage ::= 10
      ku-APReq-authenticator          KeyUsage ::= 11
      ku-EncAPRepPart                 KeyUsage ::= 12
      ku-EncKrbPrivPart               KeyUsage ::= 13
      ku-EncKrbCredPart               KeyUsage ::= 14
      ku-KrbSafe-cksum                KeyUsage ::= 15
      ku-ad-KDCIssued-cksum           KeyUsage ::= 19


      -- The following numbers are provisional...
      -- conflicts may exist elsewhere.
      ku-Ticket-cksum                 KeyUsage ::= 25
      ku-ASReq-cksum                  KeyUsage ::= 26
      ku-TGSReq-cksum                 KeyUsage ::= 27
      ku-ASRep-cksum                  KeyUsage ::= 28
      ku-TGSRep-cksum                 KeyUsage ::= 29
      ku-APReq-cksum                  KeyUsage ::= 30
      ku-APRep-cksum                  KeyUsage ::= 31
      ku-KrbCred-cksum                KeyUsage ::= 32
      ku-KrbError-cksum               KeyUsage ::= 33
      ku-KDCRep-cksum                 KeyUsage ::= 34
```

6.2. Which Key to Use

```
      -- KeyToUse identifies which key is to be used to encrypt or
      -- sign a given value.
      --
      -- Values of KeyToUse are never actually transmitted over the
      -- wire, or even used directly by the implementation in any
      -- way, as key usages are; it exists primarily to identify
      -- which key gets used for what purpose.  Thus, the specific
      -- numeric values associated with this type are irrelevant.
      KeyToUse        ::= ENUMERATED {
          -- unspecified
          key-unspecified,
          -- server long term key
          key-server,
          -- client long term key
          key-client,
          -- key selected by KDC for encryption of a KDC-REP
          key-kdc-rep,
          -- session key from ticket
          key-session,
          -- subsession key negotiated via AP-REQ/AP-REP
          key-subsession,
          ...
      }
```

6.3. EncryptionKey

The "EncryptionKey" type holds an encryption key.

```
      EncryptionKey   ::= SEQUENCE {
          keytype     [0] EType,
          keyvalue    [1] OCTET STRING
      }
```

keytype
: This "EType" identifies the encryption algorithm, described in [KCRYPTO].

keyvalue
: Contains the actual key.

6.4. EncryptedData

The "EncryptedData" type contains the encryption of another data type. The recipient uses fields within EncryptedData to determine which key to use for decryption.

```
    -- "Type" specifies which ASN.1 type is encrypted to the
    -- ciphertext in the EncryptedData.  "Keys" specifies a set of
    -- keys of which one key may be used to encrypt the data.
    -- "KeyUsages" specifies a set of key usages, one of which may
    -- be used to encrypt.
    --
    -- None of the parameters is transmitted over the wire.
    EncryptedData {
        Type, KeyToUse:Keys, KeyUsage:KeyUsages
    } ::= SEQUENCE {
        etype       [0] EType,
        kvno        [1] UInt32 OPTIONAL,
        cipher      [2] OCTET STRING (CONSTRAINED BY {
            -- must be encryption of --
            OCTET STRING (CONTAINING Type),
            -- with one of the keys -- KeyToUse:Keys,
            -- with key usage being one of --
            KeyUsage:KeyUsages
        }),
        ...
    }
```

KeyUsages
: Advisory parameter indicating which key usage to use when encrypting the ciphertext. If "KeyUsages" indicate multiple "KeyUsage" values, the detailed description of the containing message will indicate which key to use under which conditions.

Type
: Advisory parameter indicating the ASN.1 type whose DER encoding is the plaintext encrypted into the EncryptedData.

Keys
: Advisory parameter indicating which key to use to perform the encryption. If "Keys" indicate multiple "KeyToUse" values, the detailed description of the containing message will indicate which key to use under which conditions.

KeyUsages
: Advisory parameter indicating which "KeyUsage" value is used to encrypt. If "KeyUsages" indicates multiple "KeyUsage" values, the detailed description of the containing message will indicate which key usage to use under which conditions.

6.5. Checksums

Several types contain checksums (actually signatures) of data.

```
      CksumType       ::= Int32

      -- The parameters specify which key to use to produce the
      -- signature, as well as which key usage to use.  The
      -- parameters are not actually sent over the wire.
      Checksum {
          KeyToUse:Keys, KeyUsage:KeyUsages
      } ::= SEQUENCE {
          cksumtype   [0] CksumType,
          checksum    [1] OCTET STRING (CONSTRAINED BY {
              -- signed using one of the keys --
              KeyToUse:Keys,
              -- with key usage being one of --
              KeyUsage:KeyUsages
          })
      }
```

CksumType
: Integer type for assigned numbers for signature algorithms. Defined in [KCRYPTO]

Keys
: As in EncryptedData

KeyUsages
: As in EncryptedData

cksumtype
: Signature algorithm used to produce the signature.

checksum
: The actual checksum.

6.5.1. ChecksumOf

ChecksumOf is similar to "Checksum", but specifies which type is signed.

```
      -- a Checksum that must contain the checksum
      -- of a particular type
      ChecksumOf {
          Type, KeyToUse:Keys, KeyUsage:KeyUsages
      } ::= Checksum { Keys, KeyUsages } (WITH COMPONENTS {
          ...,
          checksum (CONSTRAINED BY {
              -- must be checksum of --
              OCTET STRING (CONTAINING Type)
          })
      })
```

Type
Indicates the ASN.1 type whose DER encoding is signed.

6.5.2. Signed

Signed is similar to "ChecksumOf", but contains an actual instance of the type being signed in addition to the signature.

```
-- parameterized type for wrapping authenticated plaintext
Signed {
    InnerType, KeyToUse:Keys, KeyUsage:KeyUsages
} ::= SEQUENCE {
    cksum       [0] ChecksumOf {
        InnerType, Keys, KeyUsages
    } OPTIONAL,
    inner       [1] InnerType,
    ...
}
```

7. Tickets

[A large number of items described here are duplicated in the sections describing KDC-REQ processing. Should find a way to avoid this duplication.]

A ticket binds a principal name to a session key. The important fields of a ticket are in the encrypted part.

```
-- Encrypted part of ticket
EncTicketPart   ::= CHOICE {
    rfc1510     EncTicketPart1510,
    ext         EncTicketPartExt
}


EncTicketPart1510       ::= [APPLICATION 3] SEQUENCE {
    flags               [0] TicketFlags,
    key                 [1] EncryptionKey,
    crealm              [2] RealmIA5,
    cname               [3] PrincipalNameIA5,
    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
}
```

```
   EncTicketPartExt        ::= [APPLICATION 5] SEQUENCE {
       flags               [0] TicketFlags,
       key                 [1] EncryptionKey,
       crealm              [2] RealmExt,
       cname               [3] PrincipalNameExt,
       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,
       ...,
   }
```

crealm
: This field contains the client's realm.

cname
: This field contains the client's name.

caddr
: This field lists the network addresses (if absent, all addresses are permitted) from which the ticket is valid.

Descriptions of the other fields appear in the following sections.

7.1. Timestamps

Three of the ticket timestamps may be requested from the KDC. The timestamps may differ from those requested, depending on site policy.

authtime
: The time at which the client authenticated, as recorded by the KDC.

starttime
: The earliest time when the ticket is valid. If not present, the ticket is valid starting at the authtime. This is requested as the "from" field of KDC-REQ-BODY.

endtime
: This time is requested in the "till" field of KDC-REQ-BODY. Contains the time after which the ticket will not be honored (its expiration time). Note that individual services MAY place their own limits on the life of a ticket and MAY reject tickets which have not yet expired. As such, this is really an upper bound on the expiration time for the ticket.

renew-till
This time is requested in the "rtime" field of KDC-REQ-BODY. It is only present in tickets that have the "renewable" flag set in the flags field. It indicates the maximum endtime that may be included in a renewal. It can be thought of as the absolute expiration time for the ticket, including all renewals.

7.2. Ticket Flags

A number of flags may be set in the ticket, further defining some of its capabilities. Some of these flags map to flags in a KDC request.

```
TicketFlags     ::= KerberosFlags { TicketFlagsBits }

TicketFlagsBits ::= BIT STRING {
    reserved            (0),
    forwardable         (1),
    forwarded           (2),
    proxiable           (3),
    proxy               (4),
    may-postdate        (5),
    postdated           (6),
    invalid             (7),
    renewable           (8),
    initial             (9),
    pre-authent         (10),
    hw-authent          (11),
    transited-policy-checked (12),
    ok-as-delegate      (13),
    anonymous           (14),
    cksummed-ticket     (15)
}
```

7.2.1. Flags Relating to Initial Ticket Acquisition

[adapted KCLAR 2.1.]

Several flags indicate the details of how the initial ticket was acquired.

initial
The "initial" flag indicates that a ticket was issued using the AS protocol, rather than issued based on a ticket-granting ticket. Application servers (e.g., a password-changing program) requiring a client's definite knowledge of its secret key can insist that this flag be set in any tickets they accept, thus being assured that the client's key was recently presented to the application client.

pre-authent
: The "pre-authent" flag indicates that some sort of pre-authentication was used during the AS exchange.

hw-authent
: The "hw-authent" flag indicates that some sort of hardware-based pre-authentication occurred during the AS exchange.

Both the "pre-authent" and the "hw-authent" flags may be present with or without the "initial" flag; such tickets with the "initial" flag clear are ones which are derived from initial tickets with the "pre-authent" or "hw-authent" flags set.

7.2.2. Invalid Tickets

[KCLAR 2.2.]

The "invalid" flag indicates that a ticket is invalid. Application servers MUST reject tickets which have this flag set. A postdated ticket will be issued in this form. Invalid tickets MUST be validated by the KDC before use, by presenting them to the KDC in a TGS request with the "validate" option specified. The KDC will only validate tickets after their starttime has passed. The validation is required so that postdated tickets which have been stolen before their starttime can be rendered permanently invalid (through a hot-list mechanism -- see Section 8.3.2.4).

7.2.3. OK as Delegate

[KCLAR 2.8.]

The "ok-as-delegate" flag provides a way for a KDC to communicate local realm policy to a client regarding whether the service for which the ticket is issued is trusted to accept delegated credentials. For some applications, a client may need to delegate credentials to a service to act on its behalf in contacting other services. The ability of a client to obtain a service ticket for a service conveys no information to the client about whether the service should be trusted to accept delegated credentials.

The copy of the ticket flags visible to the client may have the "ok-as-delegate" flag set to indicate to the client that the service specified in the ticket has been determined by policy of the realm to be a suitable recipient of delegation. A client can use the presence of this flag to help it make a decision whether to delegate credentials (either grant a proxy or a forwarded ticket-granting ticket) to this service. It is acceptable to ignore the value of this flag. When setting this flag, an administrator should consider the security and placement of the server on which the service will run, as well as whether the service requires the use of delegated

credentials.

7.2.4. Renewable Tickets

[adapted KCLAR 2.3.]

The "renewable" flag indicates whether the ticket may be renewed.

Renewable tickets can be used to mitigate the consequences of ticket theft. Applications may desire to hold credentials which can be valid for long periods of time. However, this can expose the credentials to potential theft for equally long periods, and those stolen credentials would be valid until the expiration time of the ticket(s). Simply using short-lived tickets and obtaining new ones periodically would require the application to have long-term access to the client's secret key, which is an even greater risk.

Renewable tickets have two "expiration times": the first is when the current instance of the ticket expires, and the second is the latest permissible value for an individual expiration time. An application client must periodically present an unexpired renewable ticket to the KDC, setting the "renew" option in the KDC request. The KDC will issue a new ticket with a new session key and a later expiration time. All other fields of the ticket are left unmodified by the renewal process. When the latest permissible expiration time arrives, the ticket expires permanently. At each renewal, the KDC MAY consult a hot-list to determine if the ticket had been reported stolen since its last renewal; it will refuse to renew such stolen tickets, and thus the usable lifetime of stolen tickets is reduced.

The "renewable" flag in a ticket is normally only interpreted by the ticket-granting service. It can usually be ignored by application servers. However, some particularly careful application servers MAY disallow renewable tickets.

If a renewable ticket is not renewed by its expiration time, the KDC will not renew the ticket. The "renewable" flag is clear by default, but a client can request it be set by setting the "renewable" option in the AS-REQ message. If it is set, then the "renew-till" field in the ticket contains the time after which the ticket may not be renewed.

7.2.5. Postdated Tickets

postdated
: indicates a ticket which has been postdated

may-postdate
: indicates that postdated tickets may be issued based on this ticket

[KCLAR 2.4.]

Applications may occasionally need to obtain tickets for use much later, e.g., a batch submission system would need tickets to be valid at the time the batch job is serviced. However, it is dangerous to hold valid tickets in a batch queue, since they will be on-line longer and more prone to theft. Postdated tickets provide a way to obtain these tickets from the KDC at job submission time, but to leave them "dormant" until they are activated and validated by a further request of the KDC. If a ticket theft were reported in the interim, the KDC would refuse to validate the ticket, and the thief would be foiled.

The "may-postdate" flag in a ticket is normally only interpreted by the TGS. It can be ignored by application servers. This flag MUST be set in a ticket-granting ticket in order for the KDC to issue a postdated ticket based on the presented ticket. It is reset by default; it MAY be requested by a client by setting the "allow-postdate" option in the AS-REQ [?also TGS-REQ?] message. This flag does not allow a client to obtain a postdated ticket-granting ticket; postdated ticket-granting tickets can only by obtained by requesting the postdating in the AS-REQ message. The life (endtime minus starttime) of a postdated ticket will be the remaining life of the ticket-granting ticket at the time of the request, unless the "renewable" option is also set, in which case it can be the full life (endtime minus starttime) of the ticket-granting ticket. The KDC MAY limit how far in the future a ticket may be postdated.

The "postdated" flag indicates that a ticket has been postdated. The application server can check the authtime field in the ticket to see when the original authentication occurred. Some services MAY choose to reject postdated tickets, or they may only accept them within a certain period after the original authentication. When the KDC issues a "postdated" ticket, it will also be marked as "invalid", so that the application client MUST present the ticket to the KDC for validation before use.

7.2.6. Proxiable and Proxy Tickets

proxy
: indicates a proxy ticket

proxiable
: indicates that proxy tickets may be issued based on this ticket

[KCLAR 2.5.]

It may be necessary for a principal to allow a service to perform an operation on its behalf. The service must be able to take on the identity of the client, but only for a particular purpose. A principal can allow a service to take on the principal's identity for

a particular purpose by granting it a proxy.

The process of granting a proxy using the "proxy" and "proxiable" flags is used to provide credentials for use with specific services. Though conceptually also a proxy, users wishing to delegate their identity in a form usable for all purposes MUST use the ticket forwarding mechanism described in the next section to forward a ticket-granting ticket.

The "proxiable" flag in a ticket is normally only interpreted by the ticket-granting service. It can be ignored by application servers. When set, this flag tells the ticket-granting server that it is OK to issue a new ticket (but not a ticket-granting ticket) with a different network address based on this ticket. This flag is set if requested by the client on initial authentication. By default, the client will request that it be set when requesting a ticket-granting ticket, and reset when requesting any other ticket.

This flag allows a client to pass a proxy to a server to perform a remote request on its behalf (e.g. a print service client can give the print server a proxy to access the client's files on a particular file server in order to satisfy a print request).

In order to complicate the use of stolen credentials, Kerberos tickets may contain a set of network addresses from which they are valid. When granting a proxy, the client MUST specify the new network address from which the proxy is to be used, or indicate that the proxy is to be issued for use from any address.

The "proxy" flag is set in a ticket by the TGS when it issues a proxy ticket. Application servers MAY check this flag and at their option they MAY require additional authentication from the agent presenting the proxy in order to provide an audit trail.

7.2.7. Forwarded and Forwardable Tickets

forwarded
: indicates a forwarded ticket

forwardable
: indicates that forwarded tickets may be issued based on this ticket

[KCLAR 2.6.]

Authentication forwarding is an instance of a proxy where the service that is granted is complete use of the client's identity. An example where it might be used is when a user logs in to a remote system and wants authentication to work from that system as if the login were local.

The "forwardable" flag in a ticket is normally only interpreted by

the ticket-granting service. It can be ignored by application

servers. The "forwardable" flag has an interpretation similar to that of the "proxiable" flag, except ticket-granting tickets may also be issued with different network addresses. This flag is reset by default, but users MAY request that it be set by setting the "forwardable" option in the AS request when they request their initial ticket-granting ticket.

This flag allows for authentication forwarding without requiring the user to enter a password again. If the flag is not set, then authentication forwarding is not permitted, but the same result can still be achieved if the user engages in the AS exchange specifying the requested network addresses and supplies a password.

The "forwarded" flag is set by the TGS when a client presents a ticket with the "forwardable" flag set and requests a forwarded ticket by specifying the "forwarded" KDC option and supplying a set of addresses for the new ticket. It is also set in all tickets issued based on tickets with the "forwarded" flag set. Application servers may choose to process "forwarded" tickets differently than non-forwarded tickets.

If addressless tickets are forwarded from one system to another, clients SHOULD still use this option to obtain a new TGT in order to have different session keys on the different systems.

7.3. Transited Realms

[KCLAR 2.7., plus new stuff]

7.4. Authorization Data

[KCLAR 5.2.6.]

```
ADType          ::= TH-id

AuthorizationData       ::= SEQUENCE OF SEQUENCE {
    ad-type             [0] ADType,
    ad-data             [1] OCTET STRING
}
```

ad-type
: This field identifies the contents of the ad-data. All negative values are reserved for local use. Non-negative values are reserved for registered use.

ad-data
: This field contains authorization data to be interpreted according to the value of the corresponding ad-type field.

Each sequence of ADType and OCTET STRING is referred to as an authorization element. Elements MAY be application specific, however, there is a common set of recursive elements that should be understood by all implementations. These elements contain other AuthorizationData, and the interpretation of the encapsulating element determines which enclosed elements must be interpreted, and which may be ignored.

Depending on the meaning of the encapsulating element, the encapsulated AuthorizationData may be ignored, interpreted as issued directly by the KDC, or be stored in a separate plaintext part of the ticket. The types of the encapsulating elements are specified as part of the Kerberos protocol because behavior based on these container elements should be understood across implementations, while other elements need only be understood by the applications which they affect.

Authorization data elements are considered critical if present in a ticket or authenticator. Unless encapsulated in a known authorization data element modifying the criticality of the elements it contains, an application server MUST reject the authentication of a client whose AP-REQ or ticket contains an unrecognized authorization data element. Authorization data is intended to restrict the use of a ticket. If the service cannot determine whether it is the target of a restriction, a security weakness may exist if the ticket can be used for that service. Authorization elements that are truly optional can be enclosed in AD-IF-RELEVANT element.

ad-type	contents of ad-data
1	DER encoding of AD-IF-RELEVANT
4	DER encoding of AD-KDCIssued
5	DER encoding of AD-AND-OR
8	DER encoding of AD-MANDATORY-FOR-KDC

7.4.1. AD-IF-RELEVANT

```
ad-if-relevant          ADType ::= int32 : 1

-- Encapsulates another AuthorizationData.
-- Intended for application servers; receiving application servers
-- MAY ignore.
AD-IF-RELEVANT          ::= AuthorizationData
```

AD elements encapsulated within the if-relevant element are intended for interpretation only by application servers that understand the

particular ad-type of the embedded element. Application servers that do not understand the type of an element embedded within the if-relevant element MAY ignore the uninterpretable element. This element promotes interoperability across implementations which may have local extensions for authorization. The ad-type for AD-IF-RELEVANT is (1).

7.4.2. AD-KDCIssued

```
-- KDC-issued privilege attributes
ad-kdcissued            ADType ::= int32 : 4

AD-KDCIssued            ::= SEQUENCE {
    ad-checksum [0] ChecksumOf {
                        AuthorizationData, { key-session },
                        { ku-ad-KDCIssued-cksum }},
    i-realm     [1] Realm OPTIONAL,
    i-sname     [2] PrincipalName OPTIONAL,
    elements    [3] AuthorizationData
}
```

ad-checksum
: A cryptographic checksum computed over the DER encoding of the AuthorizationData in the "elements" field, keyed with the session key. Its checksumtype is the mandatory checksum type for the encryption type of the session key, and its key usage value is 19.

i-realm, i-sname
: The name of the issuing principal if different from the KDC itself. This field would be used when the KDC can verify the authenticity of elements signed by the issuing principal and it allows this KDC to notify the application server of the validity of those elements.

elements
: AuthorizationData issued by the KDC.

The KDC-issued ad-data field is intended to provide a means for Kerberos credentials to embed within themselves privilege attributes and other mechanisms for positive authorization, amplifying the privileges of the principal beyond what it would have if using credentials without such an authorization-data element.

This amplification of privileges cannot be provided without this element because the definition of the authorization-data field allows elements to be added at will by the bearer of a TGT at the time that they request service tickets and elements may also be added to a delegated ticket by inclusion in the authenticator.

For KDC-issued elements this is prevented because the elements are signed by the KDC by including a checksum encrypted using the server's key (the same key used to encrypt the ticket -- or a key derived from that key). AuthorizationData encapsulated with in the AD-KDCIssued element MUST be ignored by the application server if this "signature" is not present. Further, AuthorizationData encapsulated within this element from a ticket-granting ticket MAY be interpreted by the KDC, and used as a basis according to policy for including new signed elements within derivative tickets, but they will not be copied to a derivative ticket directly. If they are copied directly to a derivative ticket by a KDC that is not aware of this element, the signature will not be correct for the application ticket elements, and the field will be ignored by the application server.

This element and the elements it encapsulates MAY be safely ignored by applications, application servers, and KDCs that do not implement this element.

The ad-type for AD-KDC-ISSUED is (4).

7.4.3. AD-AND-OR

```
ad-and-or               ADType ::= int32 : 5

AD-AND-OR               ::= SEQUENCE {
    condition-count     [0] Int32,
    elements            [1] AuthorizationData
}
```

When restrictive AD elements are encapsulated within the and-or element, the and-or element is considered satisfied if and only if at least the number of encapsulated elements specified in condition-count are satisfied. Therefore, this element MAY be used to implement an "or" operation by setting the condition-count field to 1, and it MAY specify an "and" operation by setting the condition count to the number of embedded elements. Application servers that do not implement this element MUST reject tickets that contain authorization data elements of this type.

The ad-type for AD-AND-OR is (5).

7.4.4. AD-MANDATORY-FOR-KDC

```
-- KDCs MUST interpret any AuthorizationData wrapped in this.
ad-mandatory-for-kdc            ADType ::= int32 : 8
AD-MANDATORY-FOR-KDC            ::= AuthorizationData
```

AD elements encapsulated within the mandatory-for-kdc element are to

be interpreted by the KDC. KDCs that do not understand the type of

an element embedded within the mandatory-for-kdc element MUST reject the request.

The ad-type for AD-MANDATORY-FOR-KDC is (8).

7.5. Encrypted Part of Ticket

The complete definition of the encrypted part is

```
-- Encrypted part of ticket
EncTicketPart   ::= CHOICE {
    rfc1510     EncTicketPart1510,
    ext         EncTicketPartExt
}
```

The encrypted part of the backwards-compatibility form of a ticket is:

```
EncTicketPart1510       ::= [APPLICATION 3] SEQUENCE {
    flags               [0] TicketFlags,
    key                 [1] EncryptionKey,
    crealm              [2] RealmIA5,
    cname               [3] PrincipalNameIA5,
    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
}
```

The encrypted part of the extensible form of a ticket is:

```
EncTicketPartExt        ::= [APPLICATION 5] SEQUENCE {
    flags               [0] TicketFlags,
    key                 [1] EncryptionKey,
    crealm              [2] RealmExt,
    cname               [3] PrincipalNameExt,
    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,
    ...,
}
```

7.6. Cleartext Part of Ticket

The complete definition of Ticket is:

```
Ticket          ::= CHOICE {
    rfc1510     Ticket1510,
    ext         TicketExt
}
```

The "sname" field provides the name of the target service principal in cleartext, as a hint to aid the server in choosing the correct decryption key.

The backwards-compatibility form of Ticket is:

```
Ticket1510      ::= [APPLICATION 1] SEQUENCE {
    tkt-vno     [0] INTEGER (5),
    realm       [1] RealmIA5,
    sname       [2] PrincipalNameIA5,
    enc-part    [3] EncryptedData {
        EncTicketPart1510, { key-server }, { ku-Ticket }
    }
}
```

The extensible form of Ticket is:

```
TicketExt       ::= [APPLICATION 4] Signed {
    [APPLICATION 4] SEQUENCE {
        tkt-vno     [0] INTEGER (5),
        realm       [1] RealmExt,
        sname       [2] PrincipalNameExt,
        enc-part    [3] EncryptedData {
            EncTicketPartExt, { key-server }, { ku-Ticket }
        },
        ...,
        extensions          [4] TicketExtensions OPTIONAL,
        ...
    },
    { key-ticket }, { ku-Ticket-cksum }
}
```

TicketExtensions, which may only be present in the extensible form of Ticket, are a cleartext typed hole for extension use. AuthorizationData already provides an encrypted typed hole.

```
TEType                  ::= TH-id

-- ticket extensions: for TicketExt only
TicketExtensions        ::= SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    te-type             [0] TEType,
    te-data             [1] OCTET STRING
}
```

A client will only receive an extensible Ticket if the application server supports extensibility.

8. Credential Acquisition

There are two exchanges that can be used for acquiring credentials: the AS exchange and the TGS exchange. These exchanges have many similarities, and this document describes them in parallel, noting which behaviors are specific to one type of exchange. The AS request (AS-REQ) and TGS request (TGS-REQ) are both forms of KDC requests (KDC-REQ). Likewise, the AS reply (AS-REP) and TGS reply (TGS-REP) are forms of KDC replies (KDC-REP).

The credentials acquisition protocol operates over specific transports. Additionally, specific methods exist to permit a client to discover the KDC host with which to communicate.

8.1. KDC-REQ

The KDC-REQ has a large number of fields in common between the RFC 1510 and the extensible versions. The KDC-REQ type itself is never directly encoded; it is always a part of a AS-REQ or a TGS-REQ.

```
KDC-REQ-1510    ::= SEQUENCE {
-- NOTE: first tag is [1], not [0]
    pvno        [1] INTEGER (5),
    msg-type    [2] INTEGER (  10 -- AS-REQ --
                             | 12 -- TGS-REQ -- ),
    padata      [3] SEQUENCE OF PA-DATA OPTIONAL,
    req-body    [4] KDC-REQ-BODY-1510
}


KDC-REQ-EXT     ::= SEQUENCE {
    pvno        [1] INTEGER (5),
    msg-type    [2] INTEGER (  6 -- AS-REQ --
                             | 8 -- TGS-REQ -- ),
    padata      [3] SEQUENCE (SIZE (1..MAX)) OF PA-DATA OPTIONAL,
    req-body    [4] KDC-REQ-BODY-EXT,
    ...
}
```

```
      KDC-REQ-BODY-1510       ::= SEQUENCE {
          kdc-options         [0] KDCOptions,
          cname               [1] PrincipalNameIA5 OPTIONAL
          -- Used only in AS-REQ --,

          realm               [2] RealmIA5
          -- Server's realm; also client's in AS-REQ --,

          sname               [3] PrincipalNameIA5 OPTIONAL,
          from                [4] KerberosTime OPTIONAL,
          till                [5] KerberosTime,
          rtime               [6] KerberosTime OPTIONAL,
          nonce               [7] Nonce32,
          etype               [8] SEQUENCE OF EType
          -- in preference order --,

          addresses           [9] HostAddresses OPTIONAL,
          enc-authorization-data      [10] EncryptedData {
              AuthorizationData, { key-session | key-subsession },
              { ku-TGSReqAuthData-subkey |
                ku-TGSReqAuthData-sesskey }
          } OPTIONAL,

          additional-tickets  [11] SEQUENCE OF Ticket OPTIONAL
          -- NOTE: not empty --
      }
```

```
      KDC-REQ-BODY-EXT        ::= SEQUENCE {
          kdc-options         [0] KDCOptions,
          cname               [1] PrincipalName OPTIONAL
          -- Used only in AS-REQ --,

          realm               [2] Realm
          -- Server's realm; also client's in AS-REQ --,

          sname               [3] PrincipalName OPTIONAL,
          from                [4] KerberosTime OPTIONAL,
          till                [5] KerberosTime OPTIONAL
          -- was required in rfc1510;
          -- still required for compat versions
          -- of messages --,

          rtime               [6] KerberosTime OPTIONAL,
          nonce               [7] Nonce,
          etype               [8] SEQUENCE OF EType
          -- in preference order --,

          addresses           [9] HostAddresses OPTIONAL,
          enc-authorization-data      [10] EncryptedData {
              AuthorizationData, { key-session | key-subsession },
              { ku-TGSReqAuthData-subkey |
                ku-TGSReqAuthData-sesskey }
          } OPTIONAL,

          additional-tickets  [11] SEQUENCE OF Ticket OPTIONAL
          -- NOTE: not empty --,
          ...
          lang-tags   [5] SEQUENCE (SIZE (1..MAX)) OF
                              LangTag OPTIONAL,
          ...
      }
```

Many fields of KDC-REQ-BODY correspond directly to fields of an EncTicketPart. The KDC copies most of them unchanged, provided that the requested values meet site policy.

kdc-options

These flags do not correspond directly to "flags" in EncTicketPart.

cname

This field is copied to the "cname" field in EncTicketPart. The "cname" field is required in an AS-REQ; the client places its own name here. In a TGS-REQ, the "cname" in the ticket in the AP-REQ takes precedence.

realm
: This field is the server's realm, and also holds the client's realm in an AS-REQ.

sname
: The "sname" field indicates the server's name. It may be absent in a TGS-REQ which requests user-to-user authentication, in which case the "sname" of the issued ticket will be taken from the included additional ticket.

The "from", "till", and "rtime" fields correspond to the "starttime", "endtime", and "renew-till" fields of EncTicketPart.

addresses
: This field corresponds to the "caddr" field of EncTicketPart.

enc-authorization-data
: For TGS-REQ, this field contains authorization data encrypted using either the TGT session key or the AP-REQ subsession key; the KDC may copy these into the "authorization-data" field of EncTicketPart if policy permits.

lang-tags
: Only present in the extensible messages. Specifies the set of languages which the client is willing to accept in error messages.

KDC options used in a KDC-REQ are:

```
      KDCOptions      ::= KerberosFlags { KDCOptionsBits }

      KDCOptionsBits  ::= BIT STRING {
          reserved            (0),
          forwardable         (1),
          forwarded           (2),
          proxiable           (3),
          proxy               (4),
          allow-postdate      (5),
          postdated           (6),
          unused7             (7),
          renewable           (8),
          unused9             (9),
          unused10            (10),
          unused11            (11),
          unused12            (12),
          unused13            (13),
          requestanonymous    (14),
          canonicalize        (15),
          disable-transited-check (26),
          renewable-ok        (27),
          enc-tkt-in-skey     (28),
          renew               (30),
          validate            (31)
          -- XXX need "need ticket1" flag?
      }
```

Different options apply to different phases of KDC-REQ processing.

The backwards-compatibility form of a KDC-REQ is:

```
      KDC-REQ-1510    ::= SEQUENCE {
      -- NOTE: first tag is [1], not [0]
          pvno        [1] INTEGER (5),
          msg-type    [2] INTEGER (  10 -- AS-REQ --
                                   | 12 -- TGS-REQ -- ),
          padata      [3] SEQUENCE OF PA-DATA OPTIONAL,
          req-body    [4] KDC-REQ-BODY-1510
      }
```

The extensible form of a KDC-REQ is:

```
      KDC-REQ-EXT     ::= SEQUENCE {
          pvno        [1] INTEGER (5),
          msg-type    [2] INTEGER (  6 -- AS-REQ --
                                   | 8 -- TGS-REQ -- ),
          padata      [3] SEQUENCE (SIZE (1..MAX)) OF PA-DATA OPTIONAL,
          req-body    [4] KDC-REQ-BODY-EXT,
          ...
```

```
    }
```

The backwards-compatibility form of a KDC-REQ-BODY is:

```
KDC-REQ-BODY-1510       ::= SEQUENCE {
    kdc-options         [0] KDCOptions,
    cname               [1] PrincipalNameIA5 OPTIONAL
    -- Used only in AS-REQ --,

    realm               [2] RealmIA5
    -- Server's realm; also client's in AS-REQ --,

    sname               [3] PrincipalNameIA5 OPTIONAL,
    from                [4] KerberosTime OPTIONAL,
    till                [5] KerberosTime,
    rtime               [6] KerberosTime OPTIONAL,
    nonce               [7] Nonce32,
    etype               [8] SEQUENCE OF EType
    -- in preference order --,

    addresses           [9] HostAddresses OPTIONAL,
    enc-authorization-data      [10] EncryptedData {
        AuthorizationData, { key-session | key-subsession },
        { ku-TGSReqAuthData-subkey |
          ku-TGSReqAuthData-sesskey }
    } OPTIONAL,

    additional-tickets  [11] SEQUENCE OF Ticket OPTIONAL
    -- NOTE: not empty --
}
```

The extensible form of a KDC-REQ-BODY is:

```
      KDC-REQ-BODY-EXT        ::= SEQUENCE {
          kdc-options         [0] KDCOptions,
          cname               [1] PrincipalName OPTIONAL
          -- Used only in AS-REQ --,

          realm               [2] Realm
          -- Server's realm; also client's in AS-REQ --,

          sname               [3] PrincipalName OPTIONAL,
          from                [4] KerberosTime OPTIONAL,
          till                [5] KerberosTime OPTIONAL
          -- was required in rfc1510;
          -- still required for compat versions
          -- of messages --,

          rtime               [6] KerberosTime OPTIONAL,
          nonce               [7] Nonce,
          etype               [8] SEQUENCE OF EType
          -- in preference order --,

          addresses           [9] HostAddresses OPTIONAL,
          enc-authorization-data      [10] EncryptedData {
              AuthorizationData, { key-session | key-subsession },
              { ku-TGSReqAuthData-subkey |
                ku-TGSReqAuthData-sesskey }
          } OPTIONAL,

          additional-tickets  [11] SEQUENCE OF Ticket OPTIONAL
          -- NOTE: not empty --,
          ...
          lang-tags   [5] SEQUENCE (SIZE (1..MAX)) OF
                                LangTag OPTIONAL,
          ...
      }
```

The AS-REQ is:

```
      AS-REQ  ::= CHOICE {
          rfc1510     AS-REQ-1510,
          ext         AS-REQ-EXT
      }
      AS-REQ-1510     ::= [APPLICATION 10] KDC-REQ-1510
          -- AS-REQ must include client name

      AS-REQ-EXT      ::= [APPLICATION 6] Signed {
          [APPLICATION 6] KDC-REQ-EXT, { key-client }, { ku-ASReq-cksum }
      }
      -- AS-REQ must include client name
```

A client SHOULD NOT send the extensible AS-REQ alternative to a KDC

if the client does not know that the KDC supports the extensibility

framework. A client SHOULD send the extensible AS-REQ alternative in a PA-AS-REQ PA-DATA. A KDC supporting extensibility will treat the AS-REQ contained within the PA-AS-REQ as the actual AS-REQ. [XXX what if their contents conflict?]

The TGS-REQ is:

```
TGS-REQ ::= CHOICE {
    rfc1510     TGS-REQ-1510,
    ext         TGS-REQ-EXT
}

TGS-REQ-1510    ::= [APPLICATION 12] KDC-REQ-1510

TGS-REQ-EXT     ::= [APPLICATION 8] Signed {
    [APPLICATION 8] KDC-REQ-EXT, { key-session }, { ku-TGSReq-cksum }
}
```

8.2. PA-DATA

PA-DATA have multiple uses in the Kerberos protocol. They may pre-authenticate an AS-REQ; they may also modify several of the encryption keys used in a KDC-REP. PA-DATA may also provide "hints" to the client about which long-term key (usually password-derived) to use. PA-DATA may also include "hints" about which pre-authentication mechanisms to use, or include data for input into a pre-authentication mechanism.

[XXX enumerate standard padata here]

8.3. KDC-REQ Processing

Processing of a KDC-REQ proceeds through several steps. An implementation need not perform these steps exactly as described, as long as it behaves as if the steps were performed as described. The KDC performs replay handling upon receiving the request; it then validates the request, adjusts timestamps, and selects the keys used in the reply. It copies data from the request into the issued ticket, adjusting the values to conform with its policies. The KDC then transmits the reply to the client.

8.3.1. Handling Replays

Because Kerberos can run over unreliable transports such as UDP, the KDC MUST be prepared to retransmit responses in case they are lost. If a KDC receives a request identical to one it has recently successfully processed, the KDC MUST respond with a KDC-REP message rather than a replay error. In order to reduce the amount of ciphertext given to a potential attacker, KDCs MAY send the same

response generated when the request was first handled. KDCs MUST

obey this replay behavior even if the actual transport in use is reliable. If the AP-REQ which authenticates a TGS-REQ is a replay, and the entire request is not identical to a recently successfully processed request, the KDC SHOULD return "krb-ap-err-repeat", as is appropriate for AP-REQ processing.

8.3.2. Request Validation

8.3.2.1. AS-REQ Authentication

Site policy determines whether a given client principal is required to provide some pre-authentication prior to receiving an AS-REP. Since the default reply key is typically the client's long-term password-based key, an attacker may easily request known plaintext (in the form of an AS-REP) upon which to mount a dictionary attack. Pre-authentication can limit the possibility of such an attack.

If site policy requires pre-authentication for a client principal, and no pre-authentication is provided, the KDC returns the error "kdc-err-preauth-required". Accompanying this error are "e-data" which include hints telling the client which pre-authentication mechanisms to use, or data for input to pre-authentication mechanisms (e.g., input to challenge-response systems). If pre-authentication fails, the KDC returns the error "kdc-err-preauth-failed".

[may need additional changes based on Sam's preauth draft]

8.3.2.2. TGS-REQ Authentication

A TGS-REQ has an accompanying AP-REQ, which is included in the "pa-tgs-req". The KDC MUST validate the checksum in the Authenticator of the AP-REQ, which is computed over the KDC-REQ-BODY-1510 or KDC-REQ-BODY-EXT (for TGS-REQ-1510 or TGS-REQ-EXT, respectively) of the request. [padata not signed by authenticator!] Any error from the AP-REQ validation process SHOULD be returned in a KRB-ERROR message. The service principal in the ticket of the AP-REQ may be a ticket-granting service principal, or a normal application service principal. A ticket which is not a ticket-granting ticket MUST NOT be used to issue a ticket for any service other than the one named in the ticket. In this case, the "renew", "validate", or "proxy" [?also forwarded?] option must be set in the request.

8.3.2.3. Principal Validation

If the client principal in an AS-REQ is unknown, the KDC returns the error "kdc-err-c-principal-unknown". If the server principal in a KDC-REQ is unknown, the KDC returns the error "kdc-err-s-principal-unknown".

8.3.2.4. Checking For Revoked or Invalid Tickets

[KCLAR 3.3.3.1]

Whenever a request is made to the ticket-granting server, the presented ticket(s) is(are) checked against a hot-list of tickets which have been canceled. This hot-list might be implemented by storing a range of issue timestamps for "suspect tickets"; if a presented ticket had an authtime in that range, it would be rejected. In this way, a stolen ticket-granting ticket or renewable ticket cannot be used to gain additional tickets (renewals or otherwise) once the theft has been reported to the KDC for the realm in which the server resides. Any normal ticket obtained before it was reported stolen will still be valid (because they require no interaction with the KDC), but only until their normal expiration time. If TGTs have been issued for cross-realm authentication, use of the cross-realm TGT will not be affected unless the hot-list is propagated to the KDCs for the realms for which such cross-realm tickets were issued.

If a TGS-REQ ticket has its "invalid" flag set, the KDC MUST NOT issue any ticket unless the TGS-REQ requests the "validate" option.

8.3.3. Timestamp Handling

[some aspects of timestamp handling, especially regarding postdating and renewal, are difficult to read in KCLAR... needs closer examination here]

Processing of "starttime" (requested in the "from" field) differs depending on whether the "postdated" option is set in the request. If the "postdated" option is not set, and the requested "starttime" is in the future beyond the window of acceptable clock skew, the KDC returns the error "kdc-err-cannot-postdate". If the "postdated" option is not set, and the requested "starttime" is absent or does not indicate a time in the future beyond the acceptable clock skew, the KDC sets the "starttime" to the KDC's current time. The "postdated" option MUST NOT be honored if the ticket is being requested by TGS-REQ and the "may-postdate" is not set in the TGT. Otherwise, if the "postdated" option is set, and site policy permits, the KDC sets the "starttime" as requested, and sets the "invalid" flag in the new ticket.

The "till" field is required in the RFC 1510 version of the KDC-REQ. If the "till" field is equal to "19700101000000Z" (midnight, January 1, 1970), the KDC SHOULD behave as if the "till" field were absent.

The KDC MUST NOT issue a ticket whose "starttime", "endtime", or "renew-till" time is later than the "renew-till" time of the ticket

from which it is derived.

8.3.3.1. AS-REQ Timestamp Processing

In the AS exchange, the "authtime" of a ticket is set to the local time at the KDC.

The "endtime" of the ticket will be set to the earlier of the requested "till" time and a time determined by local policy, possibly determined using factors specific to the realm or principal. For example, the expiration time MAY be set to the earliest of the following:

* the expiration time ("till" value) requested

* the ticket's start time plus the maximum allowable lifetime associated with the client principal from the authentication server's database

* the ticket's start time plus the maximum allowable lifetime associated with the server principal

* the ticket's start time plus the maximum lifetime set by the policy of the local realm

If the requested expiration time minus the start time (as determined above) is less than a site-determined minimum lifetime, an error message with code "kdc-err-never-valid" is returned. If the requested expiration time for the ticket exceeds what was determined as above, and if the "renewable-ok" option was requested, then the "renewable" flag is set in the new ticket, and the "renew-till" value is set as if the "renewable" option were requested.

If the "renewable" option has been requested or if the "renewable-ok" option has been set and a renewable ticket is to be issued, then the "renew-till" field MAY be set to the earliest of:

* its requested value

* the start time of the ticket plus the minimum of the two maximum renewable lifetimes associated with the principals' database entries

* the start time of the ticket plus the maximum renewable lifetime set by the policy of the local realm

8.3.3.2. TGS-REQ Timestamp Processing

In the TGS exchange, the KDC sets the "authtime" to that of the ticket in the AP-REQ authenticating the TGS-REQ. [?application server can print a ticket for itself with a spoofed authtime. security issues for hot-list?] [MIT implementation may change

authtime of renewed tickets; needs check...]

If the TGS-REQ has a TGT as the ticket in its AP-REQ, and the TGS-REQ requests an "endtime" (in the "till" field), then the "endtime" of the new ticket is set to the minimum of

* the requested "endtime" value,

* the "endtime" in the TGT, and

* an "endtime" determined by site policy on ticket lifetimes.

If the new ticket is a renewal, the "endtime" of the new ticket is bounded by the minimum of

* the requested "endtime" value,

* the value of the "renew-till" value of the old,

* the "starttime" of the new ticket plus the lifetime (endtime minus starttime) of the old ticket, i.e., the lifetime of the new ticket may not exceed that of the ticket being renewed [adapted from KCLAR 3.3.3.], and

* an "endtime" determined by site policy on ticket lifetimes.

When handling a TGS-REQ, a KDC MUST NOT issue a postdated ticket with a "starttime", "endtime", or "renew-till" time later than the "renew-till" time of the TGT.

8.3.4. Handling Transited Realms

The KDC checks the ticket in a TGS-REQ against site policy, unless the "disable-transited-check" option is set in the TGS-REQ. If site policy permits the transit path in the TGS-REQ ticket, the KDC sets the "transited-policy-checked" flag in the issued ticket. If the "disable-transited-check" option is set, the issued ticket will have the "transited-policy-checked" flag cleared.

8.3.5. Address Processing The requested "addresses" in the KDC-REQ are copied into the issued ticket. If the "addresses" field is absent or

empty in a TGS-REQ, the KDC copies addresses from the ticket in the TGS-REQ into the issued ticket, unless the either "forwarded" or "proxy" option is set. If the "forwarded" option is set, and the ticket in the TGS-REQ has its "forwardable" flag set, the KDC copies the addresses from the TGS-REQ, not the from TGS-REQ ticket, into the issued ticket. The KDC behaves similarly if the "proxy" option is set in the TGS-REQ and the "proxiable" flag is set in the ticket. The "proxy" option will not be honored on requests for additional ticket-granting tickets.

8.3.6. Ticket Flag Processing

Many kdc-options request that the KDC set a corresponding flag in the issued ticket. kdc-options marked with an asterisk (*) in the following table do not directly request the corresponding ticket flag and therefore require special handling.

kdc-option	ticket flag affected
forwardable	forwardable
forwarded	forwarded
proxiable	proxiable
proxy	proxy
allow-postdate	may-postdate
postdated	postdated
renewable	renewable
requestanonymous	anonymous
canonicalize	-
disable-transited-check*	transited-policy-checked
renewable-ok*	renewable
enc-tkt-in-skey	-
renew	-
validate*	invalid

forwarded
: The KDC sets the "forwarded" flag in the issued ticket if the "forwarded" option is set in the TGS-REQ and the "forwardable" flag is set in the TGS-REQ ticket.

proxy
: The KDC sets the "proxy" flag in the issued ticket if the "proxy" option is set in the TGS-REQ and the "proxiable" flag is set in the TGS-REQ ticket.

disable-transited-check
: The handling of the "disable-transited-check" kdc-option is described in Section 8.3.4.

renewable-ok
: The handling of the "renewable-ok" kdc-option is described in Section 8.3.3.1.

enc-tkt-in-skey
: This flag modifies ticket key selection to use the session key of an additional ticket included in the TGS-REQ, for the purpose of user-to-user authentication.

validate
If the "validate" kdc-option is set in a TGS-REQ, and the "starttime" has passed, the KDC will clear the "invalid" bit on the ticket before re-issuing it.

8.3.7. Key Selection

Three keys are involved in creating a KDC-REP. The reply key encrypts the encrypted part of the KDC-REP. The session key is stored in the encrypted part of the ticket, and is also present in the encrypted part of the KDC-REP so that the client can retrieve it. The ticket key is used to encrypt the ticket. These keys all have initial values for a given exchange; pre-authentication and other extension mechanisms may change the value used for any of these keys.

[again, may need changes based on Sam's preauth draft]

8.3.7.1. Reply Key and Session Key Selection

The set of encryption types which the client will understand appears in the "etype" field of KDC-REQ-BODY. The KDC limits the set of possible reply keys and the set of session key encryption types based on the "etype" field.

For the AS exchange, the reply key is initially a long-term key of the client, limited to those encryption types listed in the "etype" field. The KDC SHOULD use the first valid strong "etype" for which an encryption key is available. For the TGS exchange, the reply key is initially the subsession key of the Authenticator. If the Authenticator subsesion key is absent, the reply key is initially the session key of the ticket used to authenticate the TGS-REQ.

The session key is initially randomly generated, and has an encryption type which both the client and the server will understand. Typically, the KDC has prior knowledge of which encryption types the server will understand. It selects the first valid strong "etype" listed the request which the server also will understand.

8.3.7.2. Ticket Key Selection

The ticket key is initially the long-term key of the service. The "enc-tkt-in-skey" option requests user-to-user authentication, where the ticket encryption key of the issued ticket is set equal to the session key of the additional ticket in the request.

8.4. KDC-REP

The important parts of the KDC-REP are encrypted.

```
      EncASRepPart1510        ::= [APPLICATION 25] EncKDCRepPart1510
      EncTGSRepPart1510       ::= [APPLICATION 26] EncKDCRepPart1510

      EncASRepPartExt         ::= [APPLICATION 32] EncKDCRepPartExt
      EncTGSRepPartExt        ::= [APPLICATION 33] EncKDCRepPartExt

      EncKDCRepPart1510       ::= SEQUENCE {
          key                 [0] EncryptionKey,
          last-req            [1] LastReq,
          nonce               [2] Nonce32,
          key-expiration      [3] KerberosTime OPTIONAL,
          flags               [4] TicketFlags,
          authtime            [5] KerberosTime,
          starttime           [6] KerberosTime OPTIONAL,
          endtime             [7] KerberosTime,
          renew-till          [8] KerberosTime OPTIONAL,
          srealm              [9] RealmIA5,
          sname               [10] PrincipalNameIA5,
          caddr               [11] HostAddresses OPTIONAL
      }

      EncKDCRepPartExt        ::= SEQUENCE {
          key                 [0] EncryptionKey,
          last-req            [1] LastReq,
          nonce               [2] Nonce,
          key-expiration      [3] KerberosTime OPTIONAL,
          flags               [4] TicketFlags,
          authtime            [5] KerberosTime,
          starttime           [6] KerberosTime OPTIONAL,
          endtime             [7] KerberosTime,
          renew-till          [8] KerberosTime OPTIONAL,
          srealm              [9] Realm,
          sname               [10] PrincipalName,
          caddr               [11] HostAddresses OPTIONAL,
          ...
      }
```

Most of the fields of EncKDCRepPartCom are duplicates of the corresponding fields in the returned ticket.

```
      KDC-REP-1510 { EncPart } ::= SEQUENCE {
          pvno        [0] INTEGER (5),
          msg-type    [1] INTEGER (11 -- AS-REP.rfc1510 -- |
                                   13 -- TGS.rfc1510 -- ),
          padata      [2] SEQUENCE OF PA-DATA OPTIONAL,
          crealm      [3] RealmIA5,
          cname       [4] PrincipalNameIA5,
          ticket      [5] Ticket,

          enc-part    [6] EncryptedData {
              EncPart,
              { key-reply },
              -- maybe reach into EncryptedData in AS-REP/TGS-REP
              -- definitions to apply constraints on key usages?
              { ku-EncASRepPart -- if AS-REP -- |
                ku-EncTGSRepPart-subkey -- if TGS-REP and
                                        -- using Authenticator
                                        -- session key -- |
                ku-EncTGSRepPart-sesskey -- if TGS-REP and using
                                         -- subsession key -- }
          }
      }
```

```
      KDC-REP-EXT { EncPart } ::= SEQUENCE {
          pvno        [0] INTEGER (5),
          msg-type    [1] INTEGER (7 -- AS-REP.ext -- |
                                   9 -- TGS-REP.ext -- ),
          padata      [2] SEQUENCE OF PA-DATA OPTIONAL,
          crealm      [3] RealmExt,
          cname       [4] PrincipalNameExt,
          ticket      [5] Ticket,

          enc-part    [6] EncryptedData {
              EncPart,
              { key-reply },
              -- maybe reach into EncryptedData in AS-REP/TGS-REP
              -- definitions to apply constraints on key usages?
              { ku-EncASRepPart -- if AS-REP -- |
                ku-EncTGSRepPart-subkey -- if TGS-REP and
                                        -- using Authenticator
                                        -- session key -- |
                ku-EncTGSRepPart-sesskey -- if TGS-REP and using
                                         -- subsession key -- }
          },

          ...,
          -- In extensible version, KDC signs original request
          -- to avoid replay attacks against client.
          req-cksum   [7] ChecksumOf { CHOICE {
              as-req          AS-REQ,
              tgs-req         TGS-REQ
          }, { key-reply }, { ku-KDCRep-cksum }} OPTIONAL,
          lang-tag    [8] LangTag OPTIONAL,
          ...
      }
```

req-cksum

Signature of the original request using the reply key, to avoid replay attacks against the client, among other things. Only present in the extensible version of KDC-REP.

```
          AS-REP          ::= CHOICE {
              rfc1510     AS-REP-1510,
              ext         AS-REP-EXT
          }
          AS-REP-1510     ::= [APPLICATION 11] KDC-REP-1510
          AS-REP-EXT      ::= [APPLICATION 7] Signed {
              [APPLICATION 7] KDC-REP-EXT,
              { key-reply }, { ku-ASRep-cksum }
          }
```

```
           TGS-REP         ::= CHOICE {
               rfc1510     TGS-REP-1510,
               ext         TGS-REP-EXT
           }
           TGS-REP-1510    ::= [APPLICATION 13] KDC-REP-1510
{ EncTGSRepPart1510 }
           TGS-REP-EXT     ::= [APPLICATION 9] Signed {
               [APPLICATION 9] KDC-REP-EXT { EncTGSRepPartExt },
               { key-reply }, { ku-TGSRep-cksum }
           }
```

The extensible versions of AS-REQ and TGS-REQ are signed with the reply key, to prevent an attacker from performing a delayed denial-of-service attack by substituting the ticket.

8.5. Reply Validation

[signature verification]

8.6. IP Transports

[KCLAR 7.2]

Kerberos defines two IP transport mechanisms for the credentials acquisition protocol: UDP/IP and TCP/IP.

8.6.1. UDP/IP transport

Kerberos servers (KDCs) supporting IP transports MUST accept UDP requests and SHOULD listen for such requests on port 88 (decimal) unless specifically configured to listen on an alternative UDP port. Alternate ports MAY be used when running multiple KDCs for multiple realms on the same host.

Kerberos clients supporting IP transports SHOULD support the sending of UDP requests. Clients SHOULD use KDC discovery (Section 8.6.3) to identify the IP address and port to which they will send their request.

When contacting a KDC for a KRB_KDC_REQ request using UDP/IP transport, the client shall send a UDP datagram containing only an encoding of the request to the KDC. The KDC will respond with a reply datagram containing only an encoding of the reply message (either a KRB-ERROR or a KDC-REP) to the sending port at the sender's IP address. The response to a request made through UDP/IP transport MUST also use UDP/IP transport. If the response can not be handled using UDP (for example because it is too large), the KDC MUST return "krb-err-response-too-big", forcing the client to retry the request using the TCP transport.

8.6.2. TCP/IP transport

Kerberos servers (KDCs) supporting IP transports MUST accept TCP requests and SHOULD listen for such requests on port 88 (decimal) unless specifically configured to listen on an alternate TCP port. Alternate ports MAY be used when running multiple KDCs for multiple realms on the same host.

Clients MUST support the sending of TCP requests, but MAY choose to initially try a request using the UDP transport. Clients SHOULD use KDC discovery (Section 8.6.3) to identify the IP address and port to which they will send their request.

Implementation note: Some extensions to the Kerberos protocol will not succeed if any client or KDC not supporting the TCP transport is involved. Implementations of RFC 1510 were not required to support TCP/IP transports.

When the KDC-REQ message is sent to the KDC over a TCP stream, the response (KDC-REP or KRB-ERROR message) MUST be returned to the client on the same TCP stream that was established for the request. The KDC MAY close the TCP stream after sending a response, but MAY leave the stream open for a reasonable period of time if it expects a followup. Care must be taken in managing TCP/IP connections on the KDC to prevent denial of service attacks based on the number of open TCP/IP connections.

The client MUST be prepared to have the stream closed by the KDC at anytime after the receipt of a response. A stream closure SHOULD NOT be treated as a fatal error. Instead, if multiple exchanges are required (e.g., certain forms of pre-authentication) the client may need to establish a new connection when it is ready to send subsequent messages. A client MAY close the stream after receiving a response, and SHOULD close the stream if it does not expect to send followup messages.

A client MAY send multiple requests before receiving responses, though it must be prepared to handle the connection being closed after the first response.

Each request (KDC-REQ) and response (KDC-REP or KRB-ERROR) sent over the TCP stream is preceded by the length of the request as 4 octets in network byte order. The high bit of the length is reserved for future expansion and MUST currently be set to zero. If a KDC that does not understand how to interpret a set high bit of the length encoding receives a request with the high order bit of the length set, it MUST return a KRB-ERROR message with the error "krb-err-field-toolong" and MUST close the TCP stream.

If multiple requests are sent over a single TCP connection, and the KDC sends multiple responses, the KDC is not required to send the

responses in the order of the corresponding requests. This may permit some implementations to send each response as soon as it is ready even if earlier requests are still being processed (for example, waiting for a response from an external device or database).

8.6.3. KDC Discovery on IP Networks

Kerberos client implementations MUST provide a means for the client to determine the location of the Kerberos Key Distribution Centers (KDCs). Traditionally, Kerberos implementations have stored such configuration information in a file on each client machine. Experience has shown this method of storing configuration information presents problems with out-of-date information and scaling problems, especially when using cross-realm authentication. This section describes a method for using the Domain Name System [RFC 1035] for storing KDC location information.

8.6.3.1. DNS vs. Kerberos - Case Sensitivity of Realm Names

In Kerberos, realm names are case sensitive. While it is strongly encouraged that all realm names be all upper case this recommendation has not been adopted by all sites. Some sites use all lower case names and other use mixed case. DNS, on the other hand, is case insensitive for queries. Since the realm names "MYREALM", "myrealm", and "MyRealm" are all different, but resolve the same in the domain name system, it is necessary that only one of the possible combinations of upper and lower case characters be used in realm names.

8.6.3.2. DNS SRV records for KDC location

KDC location information is to be stored using the DNS SRV RR [RFC 2782]. The format of this RR is as follows:

```
_Service._Proto.Realm TTL Class SRV Priority Weight Port Target
```

The Service name for Kerberos is always "kerberos".

The Proto can be one of "udp", "tcp". If these SRV records are to be used, both "udp" and "tcp" records MUST be specified for all KDC deployments.

The Realm is the Kerberos realm that this record corresponds to. The realm MUST be a domain style realm name.

TTL, Class, SRV, Priority, Weight, and Target have the standard meaning as defined in RFC 2782.

As per RFC 2782 the Port number used for "_udp" and "_tcp" SRV records SHOULD be the value assigned to "kerberos" by the Internet

Assigned Number Authority: 88 (decimal) unless the KDC is configured

to listen on an alternate TCP port.

Implementation note: Many existing client implementations do not support KDC Discovery and are configured to send requests to the IANA assigned port (88 decimal), so it is strongly recommended that KDCs be configured to listen on that port.

8.6.3.3. KDC Discovery for Domain Style Realm Names on IP Networks

These are DNS records for a Kerberos realm EXAMPLE.COM. It has two Kerberos servers, kdc1.example.com and kdc2.example.com. Queries should be directed to kdc1.example.com first as per the specified priority. Weights are not used in these sample records.

```
_kerberos._udp.EXAMPLE.COM.   IN   SRV   0 0 88 kdc1.example.com.
_kerberos._udp.EXAMPLE.COM.   IN   SRV   1 0 88 kdc2.example.com.
_kerberos._tcp.EXAMPLE.COM.   IN   SRV   0 0 88 kdc1.example.com.
_kerberos._tcp.EXAMPLE.COM.   IN   SRV   1 0 88 kdc2.example.com.
```

9. Errors

The KRB-ERROR message is returned by the KDC if an error occurs during credentials acquisition. It may also be returned by an application server if an error occurs during authentication.

```
ErrCode ::= Int32

KRB-ERROR       ::= CHOICE {
    rfc1510     KRB-ERROR-1510,
    ext         KRB-ERROR-EXT
}
```

The extensible KRB-ERROR is only signed if there has been a key negotiated with its recipient. KRB-ERROR messages sent in response to AS-REQ messages will probably not be signed unless a preauthentication mechanism has negotiated a key. (Signing using a client's long-term key can expose ciphertext to dictionary attacks.)

```
      KRB-ERROR-1510 ::= [APPLICATION 30] SEQUENCE {
          pvno        [0] INTEGER (5),
          msg-type    [1] INTEGER (30),
          ctime       [2] KerberosTime OPTIONAL,
          cusec       [3] Microseconds OPTIONAL,
          stime       [4] KerberosTime,
          susec       [5] Microseconds,
          error-code  [6] ErrCode,
          crealm      [7] RealmIA5 OPTIONAL,
          cname       [8] PrincipalNameIA5 OPTIONAL,
          realm       [9] RealmIA5 -- Correct realm --,
          sname       [10] PrincipalNameIA5 -- Correct name --,
          e-text      [11] KerberosString OPTIONAL,
          e-data      [12] OCTET STRING OPTIONAL
      }


      KRB-ERROR-EXT ::= [APPLICATION 23] Signed {
          [APPLICATION 23] SEQUENCE{
              pvno        [0] INTEGER (5),
              msg-type    [1] INTEGER (23),
              ctime       [2] KerberosTime OPTIONAL,
              cusec       [3] Microseconds OPTIONAL,
              stime       [4] KerberosTime,
              susec       [5] Microseconds,
              error-code  [6] ErrCode,
              crealm      [7] Realm OPTIONAL,
              cname       [8] PrincipalName OPTIONAL,
              realm       [9] Realm -- Correct realm --,
              sname       [10] PrincipalName -- Correct name --,
              e-text      [11] KerberosString OPTIONAL,
              e-data      [12] OCTET STRING OPTIONAL,
              ...,
              typed-data          [13] TYPED-DATA OPTIONAL,
              nonce               [14] Nonce OPTIONAL,
              lang-tag            [15] LangTag OPTIONAL,
              ...
          }, { }, { ku-KrbError-cksum }
      }
```

ctime, cusec
: Client's time, if known from a KDC-REQ or AP-REQ.

stime, susec
: KDC or application server's current time.

error-code
: Numeric error code designating the error.

crealm, cname
: Client's realm and name, if known.

realm, sname
: server's realm and name. [XXX what if these aren't known?]

e-text
: Human-readable text providing additional details for the error.

e-data
: This field contains additional data about the error for use by the client to help it recover from or handle the error. If the "error-code" is "kdc-err-preauth-required", then the e-data field will contain an encoding of a sequence of padata fields, each corresponding to an acceptable pre-authentication method and optionally containing data for the method:

```
METHOD-DATA     ::= SEQUENCE OF PA-DATA
```

: For error codes defined in this document other than "kdc-err-preauth-required", the format and contents of the e-data field are implementation-defined. Similarly, for future error codes, the format and contents of the e-data field are implementation-defined unless specified.

lang-tag
: Indicates the language of the message in the "e-text" field. It is only present in the extensible KRB-ERROR.

nonce
: is the nonce from a KDC-REQ. It is only present in the extensible KRB-ERROR.

typed-data
: TYPED-DATA is a typed hole allowing for additional data to be returned in error conditions, since "e-data" is insufficiently flexible for some purposes. TYPED-DATA is only present in the extensible KRB-ERROR.

```
TDType ::= TH-id

TYPED-DATA      ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
    data-type   [0] TDType,
    data-value  [1] OCTET STRING OPTIONAL
    }
```

10. Session Key Exchange

Session key exchange consists of the AP-REQ and AP-REP messages. The client sends the AP-REQ message, and the service responds with an AP-REP message if mutual authentication is desired. Following session key exchange, the client and service share a secret session key, or possibly a subsesion key, which can be used to protect further communications. Additionally, the session key exchange process can establish initial sequence numbers which the client and service can use to detect replayed messages.

10.1. AP-REQ

An AP-REQ message contains a ticket and a authenticator. The authenticator is ciphertext encrypted with the session key contained in the ticket. The plaintext contents of the authenticator are:

```
-- plaintext of authenticator
Authenticator1510       ::= [APPLICATION 2] SEQUENCE  {
    authenticator-vno   [0] INTEGER (5),
    crealm              [1] RealmIA5,
    cname               [2] PrincipalNameIA5,
    cksum               [3] Checksum {{ key-session },
        { ku-Authenticator-cksum |
          ku-pa-TGSReq-cksum }} OPTIONAL,
    cusec               [4] Microseconds,
    ctime               [5] KerberosTime,
    subkey              [6] EncryptionKey OPTIONAL,
    seq-number          [7] SeqNum32 OPTIONAL,
    authorization-data  [8] AuthorizationData OPTIONAL
}

AuthenticatorExt        ::= [APPLICATION 35] SEQUENCE  {
    authenticator-vno   [0] INTEGER (5),
    crealm              [1] RealmExt,
    cname               [2] PrincipalNameExt,
    cksum               [3] Checksum {{ key-session },
        { ku-Authenticator-cksum |
          ku-pa-TGSReq-cksum }} OPTIONAL,
    cusec               [4] Microseconds,
    ctime               [5] KerberosTime,
    subkey              [6] EncryptionKey OPTIONAL,
    seq-number          [7] SeqNum OPTIONAL,
    authorization-data  [8] AuthorizationData OPTIONAL,
    ...
}
```

The complete definition of AP-REQ is:

```
AP-REQ          ::= CHOICE {
    rfc1510     AP-REQ-1510,
```

```
        ext         AP-REQ-EXT
    }
```

```
AP-REQ-1510      ::= [APPLICATION 14] SEQUENCE {
    pvno                [0] INTEGER (5),
    msg-type            [1] INTEGER (14),
    ap-options          [2] APOptions,
    ticket              [3] Ticket1510,
    authenticator       [4] EncryptedData {
        Authenticator1510,
        { key-session },
        { ku-APReq-authenticator |
          ku-pa-TGSReq-authenticator }
    }
}


AP-REQ-EXT      ::= [APPLICATION 18] Signed {
    [APPLICATION 18] SEQUENCE {
        pvno                [0] INTEGER (5),
        msg-type            [1] INTEGER (18),
        ap-options          [2] APOptions,
        ticket              [3] Ticket,
        authenticator       [4] EncryptedData {
            AuthenticatorExt,
            { key-session },
            { ku-APReq-authenticator |
              ku-pa-TGSReq-authenticator }
        },
        ...,
        extensions          [5] ApReqExtensions OPTIONAL,
        lang-tag            [6] SEQUENCE (SIZE (1..MAX))
                                OF LangTag OPTIONAL,
        ...
    }, { key-session }, { ku-APReq-cksum }
}


APOptions       ::= KerberosFlags { APOptionsBits }

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

10.2. AP-REP

An AP-REP message provides mutual authentication of the service to the client.

```
EncAPRepPart    ::= CHOICE {
    rfc1510     EncAPRepPart1510,
    ext         EncAPRepPartExt
}


EncAPRepPart1510        ::= [APPLICATION 27] SEQUENCE {
    ctime               [0] KerberosTime,
    cusec               [1] Microseconds,
    subkey              [2] EncryptionKey OPTIONAL,
    seq-number          [3] SeqNum32 OPTIONAL
}


EncAPRepPartExt         ::= [APPLICATION 31] SEQUENCE {
    ctime               [0] KerberosTime,
    cusec               [1] Microseconds,
    subkey              [2] EncryptionKey OPTIONAL,
    seq-number          [3] SeqNum OPTIONAL,
    ...,
    authorization-data  [4] AuthorizationData OPTIONAL,
    ...
}


AP-REP          ::= CHOICE {
    rfc1510     AP-REP-1510,
    ext         AP-REP-EXT
}


AP-REP-1510     ::= [APPLICATION 15] SEQUENCE {
    pvno        [0] INTEGER (5),
    msg-type    [1] INTEGER (15),
    enc-part    [2] EncryptedData {
        EncApRepPart1510,
        { key-session | key-subsession }, { ku-EncAPRepPart }}
}
```

```
AP-REP-EXT      ::= [APPLICATION 19] Signed {
    [APPLICATION 19] SEQUENCE {
        pvno        [0] INTEGER (5),
        msg-type    [1] INTEGER (19),
        enc-part    [2] EncryptedData {
            EncAPRepPartExt,
            { key-session | key-subsession }, { ku-EncAPRepPart }},
        ...,
        extensions          [3] ApRepExtensions OPTIONAL,
        ...
    }, { key-session | key-subsession }, { ku-APRep-cksum }
}
```

11. Session Key Use

Once a session key has been established, the client and server can use several kinds of messages to securely transmit data. KRB-SAFE provides integrity protection for application data, while KRB-PRIV provides confidentiality along with integrity protection. The KRB-CRED message provides a means of securely forwarding credentials from the client host to the server host.

11.1. KRB-SAFE

The KRB-SAFE message provides integrity protection for an included cleartext message.

```
KRB-SAFE        ::= CHOICE {
    rfc1510     KRB-SAFE-1510,
    ext         KRB-SAFE-EXT
}


KRB-SAFE-BODY   ::= SEQUENCE {
    user-data   [0] OCTET STRING,
    timestamp   [1] KerberosTime OPTIONAL,
    usec        [2] Microseconds OPTIONAL,
    seq-number  [3] SeqNum OPTIONAL,
    s-address   [4] HostAddress,
    r-address   [5] HostAddress OPTIONAL,
    ...         -- ASN.1 extensions must be excluded
                -- when sending to rfc1510 implementations
}
```

11.2. KRB-PRIV

The KRB-PRIV message provides confidentiality and integrity protection.

```
KRB-PRIV        ::= [APPLICATION 21] SEQUENCE {
    pvno        [0] INTEGER (5),
    msg-type    [1] INTEGER (21),
    enc-part    [3] EncryptedData {
        EncKrbPrivPart,
        { key-session | key-subsession }, { ku-EncKrbPrivPart }},
    ...
}


EncKrbPrivPart  ::= [APPLICATION 28] SEQUENCE {
    user-data   [0] OCTET STRING,
    timestamp   [1] KerberosTime OPTIONAL,
    usec        [2] Microseconds OPTIONAL,
    seq-number  [3] SeqNum OPTIONAL,
    s-address   [4] HostAddress -- sender's addr --,
    r-address   [5] HostAddress OPTIONAL -- recip's addr --,
    ...         -- ASN.1 extensions must be excluded
                -- when sending to rfc1510 implementations
}
```

11.3. KRB-CRED

The KRB-CRED message provides a means of securely transferring credentials from the client to the service.

```
KRB-CRED        ::= CHOICE {
    rfc1510     KRB-CRED-1510,
    ext         KRB-CRED-EXT

}


KRB-CRED-1510 ::= [APPLICATION 22] SEQUENCE {
    pvno        [0] INTEGER (5),
    msg-type    [1] INTEGER (22),
    tickets     [2] SEQUENCE OF Ticket,
    enc-part    [3] EncryptedData {
        EncKrbCredPart,
        { key-session | key-subsession }, { ku-EncKrbCredPart }}
}
```

```
KRB-CRED-EXT    ::= [APPLICATION 24] Signed {
    [APPLICATION 24] SEQUENCE {
        pvno        [0] INTEGER (5),
        msg-type    [1] INTEGER (24),
        tickets     [2] SEQUENCE OF Ticket,
        enc-part    [3] EncryptedData {
            EncKrbCredPart,
            { key-session | key-subsession }, { ku-EncKrbCredPart }},
        ...
    }, { key-session | key-subsession }, { ku-KrbCred-cksum }
}


EncKrbCredPart  ::= [APPLICATION 29] SEQUENCE {
    ticket-info [0] SEQUENCE OF KrbCredInfo,
    nonce       [1] Nonce OPTIONAL,
    timestamp   [2] KerberosTime OPTIONAL,
    usec        [3] Microseconds OPTIONAL,
    s-address   [4] HostAddress OPTIONAL,
    r-address   [5] HostAddress OPTIONAL
}


KrbCredInfo     ::= SEQUENCE {
    key         [0] EncryptionKey,
    prealm      [1] Realm OPTIONAL,
    pname       [2] PrincipalName OPTIONAL,
    flags       [3] TicketFlags OPTIONAL,
    authtime    [4] KerberosTime OPTIONAL,
    starttime   [5] KerberosTime OPTIONAL,
    endtime     [6] KerberosTime OPTIONAL,
    renew-till  [7] KerberosTime OPTIONAL,
    srealm      [8] Realm OPTIONAL,
    sname       [9] PrincipalName OPTIONAL,
    caddr       [10] HostAddresses OPTIONAL
}
```

12. Security Considerations

12.1. Time Synchronization

Time synchronization between the KDC and application servers is necessary to prevent acceptance of expired tickets.

Time synchronization is needed between application servers and clients to prevent replay attacks if a replay cache is being used. If negotiated subsession keys are used to encrypt application data, replay caches may not be necessary.

12.2. Replays

12.3. Principal Name Reuse

See Section 5.3.

12.4. Password Guessing

12.5. Forward Secrecy

[from KCLAR 10.; needs some rewriting]

The Kerberos protocol in its basic form does not provide perfect forward secrecy for communications. If traffic has been recorded by an eavesdropper, then messages encrypted using the KRB-PRIV message, or messages encrypted using application-specific encryption under keys exchanged using Kerberos can be decrypted if any of the user's, application server's, or KDC's key is subsequently discovered. This is because the session key used to encrypt such messages is transmitted over the network encrypted in the key of the application server, and also encrypted under the session key from the user's ticket-granting ticket when returned to the user in the TGS-REP message. The session key from the ticket-granting ticket was sent to the user in the AS-REP message encrypted in the user's secret key, and embedded in the ticket-granting ticket, which was encrypted in the key of the KDC. Application requiring perfect forward secrecy must exchange keys through mechanisms that provide such assurance, but may use Kerberos for authentication of the encrypted channel established through such other means.

12.6. Authorization

As an authentication service, Kerberos provides a means of verifying the identity of principals on a network. Kerberos does not, by itself, provide authorization. Applications SHOULD NOT accept the mere issuance of a service ticket by the Kerberos server as granting authority to use the service.

12.7. Login Authentication

Some applications, particularly those which provide login access when provided with a password, SHOULD NOT treat successful acquisition of credentials as sufficient proof of the user's identity. An attacker posing as a user could generate an illegitimate KDC-REP message which decrypts properly. To authenticate a user logging on to a local system, the credentials obtained SHOULD be used in a TGS exchange to obtain credentials for a local service. Successful use of those credentials to authenticate to the local service assures that the initially obtained credentials are from a valid KDC.

13. IANA Considerations

[needs more work]

Each use of Int32 in this document defines a number space. [XXX enumerate these] Negative numbers are reserved for private use; local and experimental extensions should use these values. Zero is reserved and may not be assigned.

Typed hole contents may be identified by either Int32 values or by RELATIVE-OID values. Since RELATIVE-OIDs define a hierarchical namespace, assignments to the top level of the RELATIVE-OID space may be made on a first-come, first-served basis.

14. Acknowledgments

Much of the text here is adapted from draft-ietf-krb-wg-kerberos-clarifications-07. The description of the general form of the extensibility framework was derived from text by Sam Hartman. Some text concerning internationalization of internationalized domain names in principal names and realm names was contributed by Jeffrey Altman and Jeffrey Hutzelman.

Appendices

A. ASN.1 Module (Normative)

```
KerberosV5Spec3 {
    iso(1) identified-organization(3) dod(6) internet(1)
    security(5) kerberosV5(2) modules(4) krb5spec3(4)
} DEFINITIONS EXPLICIT TAGS ::= BEGIN


-- OID arc for KerberosV5
--
-- This OID may be used to identify Kerberos protocol messages
-- encapsulated in other protocols.
--
-- This OID also designates the OID arc for KerberosV5-related
-- OIDs.
--
-- NOTE: RFC 1510 had an incorrect value (5) for "dod" in its
-- OID.
id-krb5         OBJECT IDENTIFIER ::= {
    iso(1) identified-organization(3) dod(6) internet(1)
    security(5) kerberosV5(2)
}
```

```
      -- top-level type
      --
      -- Applications should not directly reference any types
      -- other than KRB-PDU and its component types.
      --
      KRB-PDU         ::= CHOICE {
          ticket      Ticket,
          as-req      AS-REQ,
          as-rep      AS-REP,
          tgs-req     TGS-REQ,
          tgs-rep     TGS-REP,
          ap-req      AP-REQ,
          ap-rep      AP-REP,
          krb-safe    KRB-SAFE,
          krb-priv    KRB-PRIV,
          krb-cred    KRB-CRED,
          tgt-req     TGT-REQ,
          krb-error   KRB-ERROR,
          ...
      }


      --
      -- *** basic types
      --


      -- signed values representable in 32 bits
      --
      -- These are often used as assigned numbers for various things.
      Int32           ::= INTEGER (-2147483648..2147483647)


      -- Typed hole identifiers
      TH-id           ::= CHOICE {
          int32               Int32,
          rel-oid             RELATIVE-OID
      }


      -- unsigned 32 bit values
      UInt32          ::= INTEGER (0..4294967295)


      -- unsigned 64 bit values
      UInt64          ::= INTEGER (0..18446744073709551615)


      -- sequence numbers
      SeqNum          ::= UInt64
```

```
      -- nonces
      Nonce           ::= UInt64


      -- microseconds
      Microseconds    ::= INTEGER (0..999999)


      KerberosTime    ::= GeneralizedTime (CONSTRAINED BY {
                              -- MUST NOT include fractional seconds
      })


      -- used for names and for error messages
      KerberosString  ::= CHOICE {
          ia5         GeneralString (IA5String),
          utf8        UTF8String,
          ...         -- no extension may be sent
                      -- to an rfc1510 implementation --
      }


      -- IA5 choice only; useful for constraints
      KerberosStringIA5       ::= KerberosString
          (WITH COMPONENTS { ia5 PRESENT })

      -- IA5 excluded; useful for constraints
      KerberosStringExt       ::= KerberosString
          (WITH COMPONENTS { ia5 ABSENT })


      -- used for language tags
      LangTag ::= PrintableString
          (FROM ("A".."Z" | "a".."z" | "0".."9" | "-"))
```

```
      -- assigned numbers for name types (used in principal names)
      NameType        ::= Int32

      -- Name type not known
      nt-unknown              NameType ::= 0
      -- Just the name of the principal as in DCE, or for users
      nt-principal            NameType ::= 1
      -- Service and other unique instance (krbtgt)
      nt-srv-inst             NameType ::= 2
      -- Service with host name as instance (telnet, rcommands)
      nt-srv-hst              NameType ::= 3
      -- Service with host as remaining components
      nt-srv-xhst             NameType ::= 4
      -- Unique ID
      nt-uid                  NameType ::= 5
      -- Encoded X.509 Distingished name [RFC 2253]
      nt-x500-principal       NameType ::= 6
      -- Name in form of SMTP email name (e.g. user@foo.com)
      nt-smtp-name            NameType ::= 7
      -- Enterprise name - may be mapped to principal name
      nt-enterprise           NameType ::= 10


      PrincipalName { StrType }       ::= SEQUENCE {
          name-type   [0] NameType,
          -- May have zero elements, or individual elements may be
          -- zero-length, but this is NOT RECOMMENDED.
          name-string [1] SEQUENCE OF KerberosString (StrType)
      }


      -- IA5 only
      PrincipalNameIA5 ::= PrincipalName { KerberosStringIA5 }
      -- IA5 excluded
      PrincipalNameExt ::= PrincipalName { KerberosStringExt }
      -- Either one?
      PrincipalNameEither ::= PrincipalName { KerberosString }


      Realm { StrType }       ::= KerberosString (StrType)

      -- IA5 only
      RealmIA5                ::= Realm { KerberosStringIA5 }

      -- IA5 excluded
      RealmExt                ::= Realm { KerberosStringExt }

      -- Either
      RealmEither             ::= Realm { KerberosString }
```

```
      KerberosFlags { NamedBits } ::= BIT STRING (SIZE (32..MAX))
          (CONSTRAINED BY {
          -- MUST be a valid value of -- NamedBits
          -- but if the value to be sent would be truncated to shorter
          -- than 32 bits according to DER, the value MUST be padded
          -- with trailing zero bits to 32 bits.  Otherwise, no
          -- trailing zero bits may be present.

      })


      AddrType        ::= Int32

      HostAddress     ::= SEQUENCE  {
          addr-type   [0] AddrType,
          address     [1] OCTET STRING
      }

      -- NOTE: HostAddresses is always used as an OPTIONAL field and
      -- should not be a zero-length SEQUENCE OF.
      --
      -- The extensible messages explicitly constrain this to be
      -- non-empty.
      HostAddresses   ::= SEQUENCE OF HostAddress


      --
      -- *** crypto-related types and assignments
      --
```

```
      -- Assigned numbers denoting encryption mechanisms.
      EType ::= Int32

      -- assigned numbers for encryption schemes
      et-des-cbc-crc                  EType ::= 1
      et-des-cbc-md4                  EType ::= 2
      et-des-cbc-md5                  EType ::= 3
      --     [reserved]                        4
      et-des3-cbc-md5                 EType ::= 5
      --     [reserved]                        6
      et-des3-cbc-sha1                EType ::= 7
      et-dsaWithSHA1-CmsOID           EType ::= 9
      et-md5WithRSAEncryption-CmsOID  EType ::= 10
      et-sha1WithRSAEncryption-CmsOID EType ::= 11
      et-rc2CBC-EnvOID                EType ::= 12
      et-rsaEncryption-EnvOID         EType ::= 13
      et-rsaES-OAEP-ENV-OID           EType ::= 14
      et-des-ede3-cbc-Env-OID         EType ::= 15
      et-des3-cbc-sha1-kd             EType ::= 16
      -- AES
      et-aes128-cts-hmac-sha1-96      EType ::= 17
      -- AES
      et-aes256-cts-hmac-sha1-96      EType ::= 18
      -- Microsoft
      et-rc4-hmac                     EType ::= 23
      -- Microsoft
      et-rc4-hmac-exp                 EType ::= 24
      -- opaque; PacketCable
      et-subkey-keymaterial           EType ::= 65
```

```
      -- Assigned numbers denoting key usages.
      KeyUsage ::= UInt32

      --
      -- Actual identifier names are provisional and subject to
      -- change.
      --
      ku-pa-enc-ts                    KeyUsage ::= 1
      ku-Ticket                       KeyUsage ::= 2
      ku-EncASRepPart                 KeyUsage ::= 3
      ku-TGSReqAuthData-sesskey       KeyUsage ::= 4
      ku-TGSReqAuthData-subkey        KeyUsage ::= 5
      ku-pa-TGSReq-cksum              KeyUsage ::= 6
      ku-pa-TGSReq-authenticator      KeyUsage ::= 7
      ku-EncTGSRepPart-sesskey        KeyUsage ::= 8
      ku-EncTGSRepPart-subkey         KeyUsage ::= 9
      ku-Authenticator-cksum          KeyUsage ::= 10
      ku-APReq-authenticator          KeyUsage ::= 11
      ku-EncAPRepPart                 KeyUsage ::= 12
      ku-EncKrbPrivPart               KeyUsage ::= 13
      ku-EncKrbCredPart               KeyUsage ::= 14
      ku-KrbSafe-cksum                KeyUsage ::= 15
      ku-ad-KDCIssued-cksum           KeyUsage ::= 19


      -- The following numbers are provisional...
      -- conflicts may exist elsewhere.
      ku-Ticket-cksum                 KeyUsage ::= 25
      ku-ASReq-cksum                  KeyUsage ::= 26
      ku-TGSReq-cksum                 KeyUsage ::= 27
      ku-ASRep-cksum                  KeyUsage ::= 28
      ku-TGSRep-cksum                 KeyUsage ::= 29
      ku-APReq-cksum                  KeyUsage ::= 30
      ku-APRep-cksum                  KeyUsage ::= 31
      ku-KrbCred-cksum                KeyUsage ::= 32
      ku-KrbError-cksum               KeyUsage ::= 33
      ku-KDCRep-cksum                 KeyUsage ::= 34
```

```
      -- KeyToUse identifies which key is to be used to encrypt or
      -- sign a given value.
      --
      -- Values of KeyToUse are never actually transmitted over the
      -- wire, or even used directly by the implementation in any
      -- way, as key usages are; it exists primarily to identify
      -- which key gets used for what purpose.  Thus, the specific
      -- numeric values associated with this type are irrelevant.
      KeyToUse        ::= ENUMERATED {
          -- unspecified
          key-unspecified,
          -- server long term key
          key-server,
          -- client long term key
          key-client,
          -- key selected by KDC for encryption of a KDC-REP
          key-kdc-rep,
          -- session key from ticket
          key-session,
          -- subsession key negotiated via AP-REQ/AP-REP
          key-subsession,
          ...
      }


      EncryptionKey   ::= SEQUENCE {
          keytype     [0] EType,
          keyvalue    [1] OCTET STRING
      }
```

```
      -- "Type" specifies which ASN.1 type is encrypted to the
      -- ciphertext in the EncryptedData.  "Keys" specifies a set of
      -- keys of which one key may be used to encrypt the data.
      -- "KeyUsages" specifies a set of key usages, one of which may
      -- be used to encrypt.
      --
      -- None of the parameters is transmitted over the wire.
      EncryptedData {
          Type, KeyToUse:Keys, KeyUsage:KeyUsages
      } ::= SEQUENCE {
          etype       [0] EType,
          kvno        [1] UInt32 OPTIONAL,
          cipher      [2] OCTET STRING (CONSTRAINED BY {
              -- must be encryption of --
              OCTET STRING (CONTAINING Type),
              -- with one of the keys -- KeyToUse:Keys,
              -- with key usage being one of --
              KeyUsage:KeyUsages
          }),
          ...
      }



      CksumType       ::= Int32

      -- The parameters specify which key to use to produce the
      -- signature, as well as which key usage to use.  The
      -- parameters are not actually sent over the wire.
      Checksum {
          KeyToUse:Keys, KeyUsage:KeyUsages
      } ::= SEQUENCE {
          cksumtype   [0] CksumType,
          checksum    [1] OCTET STRING (CONSTRAINED BY {
              -- signed using one of the keys --
              KeyToUse:Keys,
              -- with key usage being one of --
              KeyUsage:KeyUsages
          })
      }
```

```
      -- a Checksum that must contain the checksum
      -- of a particular type
      ChecksumOf {
          Type, KeyToUse:Keys, KeyUsage:KeyUsages
      } ::= Checksum { Keys, KeyUsages } (WITH COMPONENTS {
          ...,
          checksum (CONSTRAINED BY {
              -- must be checksum of --
              OCTET STRING (CONTAINING Type)
          })
      })


      -- parameterized type for wrapping authenticated plaintext
      Signed {
          InnerType, KeyToUse:Keys, KeyUsage:KeyUsages
      } ::= SEQUENCE {
          cksum       [0] ChecksumOf {
              InnerType, Keys, KeyUsages
          } OPTIONAL,
          inner       [1] InnerType,
          ...
      }


      --
      -- *** Tickets
      --


      Ticket          ::= CHOICE {
          rfc1510     Ticket1510,
          ext         TicketExt
      }


      Ticket1510      ::= [APPLICATION 1] SEQUENCE {
          tkt-vno     [0] INTEGER (5),
          realm       [1] RealmIA5,
          sname       [2] PrincipalNameIA5,
          enc-part    [3] EncryptedData {
              EncTicketPart1510, { key-server }, { ku-Ticket }
          }
      }
```

```
TicketExt       ::= [APPLICATION 4] Signed {
    [APPLICATION 4] SEQUENCE {
        tkt-vno     [0] INTEGER (5),
        realm       [1] RealmExt,
        sname       [2] PrincipalNameExt,
        enc-part    [3] EncryptedData {
            EncTicketPartExt, { key-server }, { ku-Ticket }
        },
        ...,
        extensions          [4] TicketExtensions OPTIONAL,
        ...
    },
    { key-ticket }, { ku-Ticket-cksum }
}


-- Encrypted part of ticket
EncTicketPart   ::= CHOICE {
    rfc1510     EncTicketPart1510,
    ext         EncTicketPartExt
}


EncTicketPart1510       ::= [APPLICATION 3] SEQUENCE {
    flags               [0] TicketFlags,
    key                 [1] EncryptionKey,
    crealm              [2] RealmIA5,
    cname               [3] PrincipalNameIA5,
    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
}
```

```
      EncTicketPartExt        ::= [APPLICATION 5] SEQUENCE {
          flags               [0] TicketFlags,
          key                 [1] EncryptionKey,
          crealm              [2] RealmExt,
          cname               [3] PrincipalNameExt,
          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,
          ...,
      }


      --
      -- *** Authorization Data
      --


      ADType          ::= TH-id

      AuthorizationData       ::= SEQUENCE OF SEQUENCE {
          ad-type             [0] ADType,
          ad-data             [1] OCTET STRING
      }


      ad-if-relevant          ADType ::= int32 : 1

      -- Encapsulates another AuthorizationData.
      -- Intended for application servers; receiving application servers
      -- MAY ignore.
      AD-IF-RELEVANT          ::= AuthorizationData


      -- KDC-issued privilege attributes
      ad-kdcissued            ADType ::= int32 : 4

      AD-KDCIssued            ::= SEQUENCE {
          ad-checksum [0] ChecksumOf {
                              AuthorizationData, { key-session },
                              { ku-ad-KDCIssued-cksum }},
          i-realm     [1] Realm OPTIONAL,
          i-sname     [2] PrincipalName OPTIONAL,
          elements    [3] AuthorizationData
      }
```

```
      ad-and-or               ADType ::= int32 : 5

      AD-AND-OR               ::= SEQUENCE {
          condition-count     [0] Int32,
          elements            [1] AuthorizationData
      }


      -- KDCs MUST interpret any AuthorizationData wrapped in this.
      ad-mandatory-for-kdc            ADType ::= int32 : 8
      AD-MANDATORY-FOR-KDC            ::= AuthorizationData


      ad-initial-verified-cas         ADType ::= int32 : 9


      TrType                  ::= TH-id -- must be registered

      -- encoded Transited field
      TransitedEncoding       ::= SEQUENCE {
          tr-type     [0] TrType,
          contents    [1] OCTET STRING
      }


      TEType                  ::= TH-id

      -- ticket extensions: for TicketExt only
      TicketExtensions        ::= SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
          te-type             [0] TEType,
          te-data             [1] OCTET STRING
      }
```

```
      TicketFlags     ::= KerberosFlags { TicketFlagsBits }

      TicketFlagsBits ::= BIT STRING {
          reserved            (0),
          forwardable         (1),
          forwarded           (2),
          proxiable           (3),
          proxy               (4),
          may-postdate        (5),
          postdated           (6),
          invalid             (7),
          renewable           (8),
          initial             (9),
          pre-authent         (10),
          hw-authent          (11),
          transited-policy-checked (12),
          ok-as-delegate      (13),
          anonymous           (14),
          cksummed-ticket     (15)
      }


      --
      -- *** KDC protocol
      --


      AS-REQ  ::= CHOICE {
          rfc1510     AS-REQ-1510,
          ext         AS-REQ-EXT
      }
      AS-REQ-1510     ::= [APPLICATION 10] KDC-REQ-1510
          -- AS-REQ must include client name

      AS-REQ-EXT      ::= [APPLICATION 6] Signed {
          [APPLICATION 6] KDC-REQ-EXT, { key-client }, { ku-ASReq-cksum }
      }
      -- AS-REQ must include client name


      TGS-REQ ::= CHOICE {
          rfc1510     TGS-REQ-1510,
          ext         TGS-REQ-EXT
      }

      TGS-REQ-1510    ::= [APPLICATION 12] KDC-REQ-1510

      TGS-REQ-EXT     ::= [APPLICATION 8] Signed {
          [APPLICATION 8] KDC-REQ-EXT, { key-session }, { ku-TGSReq-cksum }
      }
```

```
KDC-REQ-1510    ::= SEQUENCE {
-- NOTE: first tag is [1], not [0]
    pvno        [1] INTEGER (5),
    msg-type    [2] INTEGER (  10 -- AS-REQ --
                             | 12 -- TGS-REQ -- ),
    padata      [3] SEQUENCE OF PA-DATA OPTIONAL,
    req-body    [4] KDC-REQ-BODY-1510
}


KDC-REQ-EXT     ::= SEQUENCE {
    pvno        [1] INTEGER (5),
    msg-type    [2] INTEGER (  6 -- AS-REQ --
                             | 8 -- TGS-REQ -- ),
    padata      [3] SEQUENCE (SIZE (1..MAX)) OF PA-DATA OPTIONAL,
    req-body    [4] KDC-REQ-BODY-EXT,
    ...
}


KDC-REQ-BODY-1510       ::= SEQUENCE {
    kdc-options         [0] KDCOptions,
    cname               [1] PrincipalNameIA5 OPTIONAL
    -- Used only in AS-REQ --,

    realm               [2] RealmIA5
    -- Server's realm; also client's in AS-REQ --,

    sname               [3] PrincipalNameIA5 OPTIONAL,
    from                [4] KerberosTime OPTIONAL,
    till                [5] KerberosTime,
    rtime               [6] KerberosTime OPTIONAL,
    nonce               [7] Nonce32,
    etype               [8] SEQUENCE OF EType
    -- in preference order --,

    addresses           [9] HostAddresses OPTIONAL,
    enc-authorization-data      [10] EncryptedData {
        AuthorizationData, { key-session | key-subsession },
        { ku-TGSReqAuthData-subkey |
          ku-TGSReqAuthData-sesskey }
    } OPTIONAL,

    additional-tickets  [11] SEQUENCE OF Ticket OPTIONAL
    -- NOTE: not empty --
}
```

```
      KDC-REQ-BODY-EXT        ::= SEQUENCE {
          kdc-options         [0] KDCOptions,
          cname               [1] PrincipalName OPTIONAL
          -- Used only in AS-REQ --,

          realm               [2] Realm
          -- Server's realm; also client's in AS-REQ --,

          sname               [3] PrincipalName OPTIONAL,
          from                [4] KerberosTime OPTIONAL,
          till                [5] KerberosTime OPTIONAL
          -- was required in rfc1510;
          -- still required for compat versions
          -- of messages --,

          rtime               [6] KerberosTime OPTIONAL,
          nonce               [7] Nonce,
          etype               [8] SEQUENCE OF EType
          -- in preference order --,

          addresses           [9] HostAddresses OPTIONAL,
          enc-authorization-data      [10] EncryptedData {
              AuthorizationData, { key-session | key-subsession },
              { ku-TGSReqAuthData-subkey |
                ku-TGSReqAuthData-sesskey }
          } OPTIONAL,

          additional-tickets  [11] SEQUENCE OF Ticket OPTIONAL
          -- NOTE: not empty --,
          ...
          lang-tags   [5] SEQUENCE (SIZE (1..MAX)) OF
                              LangTag OPTIONAL,
          ...
      }
```

```
      KDCOptions      ::= KerberosFlags { KDCOptionsBits }

      KDCOptionsBits  ::= BIT STRING {
          reserved            (0),
          forwardable         (1),
          forwarded           (2),
          proxiable           (3),
          proxy               (4),
          allow-postdate      (5),
          postdated           (6),
          unused7             (7),
          renewable           (8),
          unused9             (9),
          unused10            (10),
          unused11            (11),
          unused12            (12),
          unused13            (13),
          requestanonymous    (14),
          canonicalize        (15),
          disable-transited-check (26),
          renewable-ok        (27),
          enc-tkt-in-skey     (28),
          renew               (30),
          validate            (31)
          -- XXX need "need ticket1" flag?
      }


      AS-REP          ::= CHOICE {
          rfc1510     AS-REP-1510,
          ext         AS-REP-EXT
      }
      AS-REP-1510     ::= [APPLICATION 11] KDC-REP-1510
      AS-REP-EXT      ::= [APPLICATION 7] Signed {
          [APPLICATION 7] KDC-REP-EXT,
          { key-reply }, { ku-ASRep-cksum }
      }


      TGS-REP         ::= CHOICE {
          rfc1510     TGS-REP-1510,
          ext         TGS-REP-EXT
      }
      TGS-REP-1510    ::= [APPLICATION 13] KDC-REP-1510 { EncTGSRepPart1510 }
      TGS-REP-EXT     ::= [APPLICATION 9] Signed {
          [APPLICATION 9] KDC-REP-EXT { EncTGSRepPartExt },
          { key-reply }, { ku-TGSRep-cksum }
      }
```

```
      KDC-REP-1510 { EncPart } ::= SEQUENCE {
          pvno        [0] INTEGER (5),
          msg-type    [1] INTEGER (11 -- AS-REP.rfc1510 -- |
                                   13 -- TGS.rfc1510 -- ),
          padata      [2] SEQUENCE OF PA-DATA OPTIONAL,
          crealm      [3] RealmIA5,
          cname       [4] PrincipalNameIA5,
          ticket      [5] Ticket,

          enc-part    [6] EncryptedData {
              EncPart,
              { key-reply },
              -- maybe reach into EncryptedData in AS-REP/TGS-REP
              -- definitions to apply constraints on key usages?
              { ku-EncASRepPart -- if AS-REP -- |
                ku-EncTGSRepPart-subkey -- if TGS-REP and
                                        -- using Authenticator
                                        -- session key -- |
                ku-EncTGSRepPart-sesskey -- if TGS-REP and using
                                         -- subsession key -- }
          }
      }
```

```
      KDC-REP-EXT { EncPart } ::= SEQUENCE {
          pvno        [0] INTEGER (5),
          msg-type    [1] INTEGER (7 -- AS-REP.ext -- |
                                   9 -- TGS-REP.ext -- ),
          padata      [2] SEQUENCE OF PA-DATA OPTIONAL,
          crealm      [3] RealmExt,
          cname       [4] PrincipalNameExt,
          ticket      [5] Ticket,

          enc-part    [6] EncryptedData {
              EncPart,
              { key-reply },
              -- maybe reach into EncryptedData in AS-REP/TGS-REP
              -- definitions to apply constraints on key usages?
              { ku-EncASRepPart -- if AS-REP -- |
                ku-EncTGSRepPart-subkey -- if TGS-REP and
                                        -- using Authenticator
                                        -- session key -- |
                ku-EncTGSRepPart-sesskey -- if TGS-REP and using
                                         -- subsession key -- }
          },

          ...,
          -- In extensible version, KDC signs original request
          -- to avoid replay attacks against client.
          req-cksum   [7] ChecksumOf { CHOICE {
              as-req          AS-REQ,
              tgs-req         TGS-REQ
          }, { key-reply }, { ku-KDCRep-cksum }} OPTIONAL,
          lang-tag    [8] LangTag OPTIONAL,
          ...
      }
```

```
      EncASRepPart1510        ::= [APPLICATION 25] EncKDCRepPart1510
      EncTGSRepPart1510       ::= [APPLICATION 26] EncKDCRepPart1510

      EncASRepPartExt         ::= [APPLICATION 32] EncKDCRepPartExt
      EncTGSRepPartExt        ::= [APPLICATION 33] EncKDCRepPartExt

      EncKDCRepPart1510       ::= SEQUENCE {
          key                 [0] EncryptionKey,
          last-req            [1] LastReq,
          nonce               [2] Nonce32,
          key-expiration      [3] KerberosTime OPTIONAL,
          flags               [4] TicketFlags,
          authtime            [5] KerberosTime,
          starttime           [6] KerberosTime OPTIONAL,
          endtime             [7] KerberosTime,
          renew-till          [8] KerberosTime OPTIONAL,
          srealm              [9] RealmIA5,
          sname               [10] PrincipalNameIA5,
          caddr               [11] HostAddresses OPTIONAL
      }

      EncKDCRepPartExt        ::= SEQUENCE {
          key                 [0] EncryptionKey,
          last-req            [1] LastReq,
          nonce               [2] Nonce,
          key-expiration      [3] KerberosTime OPTIONAL,
          flags               [4] TicketFlags,
          authtime            [5] KerberosTime,
          starttime           [6] KerberosTime OPTIONAL,
          endtime             [7] KerberosTime,
          renew-till          [8] KerberosTime OPTIONAL,
          srealm              [9] Realm,
          sname               [10] PrincipalName,
          caddr               [11] HostAddresses OPTIONAL,
          ...
      }


      LRType          ::=     TH-id
      LastReq         ::=     SEQUENCE OF SEQUENCE {
          lr-type     [0] LRType,
          lr-value    [1] KerberosTime
      }


      --
      -- *** preauth
      --
```

```
      PaDataType      ::= TH-id
      PaDataOID       ::= RELATIVE-OID

      PA-DATA ::= SEQUENCE {
          -- NOTE: first tag is [1], not [0]
          padata-type         [1] PaDataType,
          padata-value        [2] OCTET STRING
      }


      -- AP-REQ authenticating a TGS-REQ
      pa-tgs-req              PaDataType ::= int32 : 1
      PA-TGS-REQ              ::= AP-REQ


      -- Encrypted timestamp preauth
      -- Encryption key used is client's long-term key.
      pa-enc-timestamp        PaDataType ::= int32 : 2

      PA-ENC-TIMESTAMP ::= EncryptedData {
          PA-ENC-TS-ENC, { key-client }, { ku-pa-enc-ts }
      }

      PA-ENC-TS-ENC           ::= SEQUENCE {
              patimestamp     [0] KerberosTime -- client's time --,
              pausec          [1] Microseconds OPTIONAL
      }


      -- Hints returned in AS-REP or KRB-ERROR to help client
      -- choose a password-derived key, either for preauthentication
      -- or for decryption of the reply.
      pa-etype-info           PaDataType ::= int32 : 11

      ETYPE-INFO              ::= SEQUENCE OF ETYPE-INFO-ENTRY

      ETYPE-INFO-ENTRY        ::= SEQUENCE {
              etype           [0] EType,
              salt            [1] OCTET STRING OPTIONAL
      }
```

```
      -- Similar to etype-info, but with parameters provided for
      -- the string-to-key function.
      pa-etype-info2          PaDataType ::= int32 : 19

      ETYPE-INFO2             ::= SEQUENCE (SIZE (1..MAX))
                                      OF ETYPE-INFO-ENTRY

      ETYPE-INFO2-ENTRY       ::= SEQUENCE {
              etype           [0] EType,
              salt            [1] KerberosString OPTIONAL,
              s2kparams       [2] OCTET STRING OPTIONAL
      }


      -- Obsolescent.  Salt for client long-term key
      -- Its character encoding is unspecified.
      pa-pw-salt              PaDataType ::= int32 : 3

      -- The "padata-value" does not encode an ASN.1 type.
      -- Instead, "padata-value" must consist of the salt string to
      -- be used by the client, in an unspecified character
      -- encoding.


      -- An extensible AS-REQ may be sent as a padata in a
      -- non-extensible AS-REQ to allow for backwards compatibility.
      pa-as-req               PaDataType ::= int32 : 42 -- provisional
      PA-AS-REQ               ::= AS-REQ (WITH COMPONENTS ext)


      --
      -- *** Session key exchange
      --


      AP-REQ          ::= CHOICE {
          rfc1510     AP-REQ-1510,
          ext         AP-REQ-EXT
      }
```

```
AP-REQ-1510      ::= [APPLICATION 14] SEQUENCE {
    pvno                [0] INTEGER (5),
    msg-type            [1] INTEGER (14),
    ap-options          [2] APOptions,
    ticket              [3] Ticket1510,
    authenticator       [4] EncryptedData {
        Authenticator1510,
        { key-session },
        { ku-APReq-authenticator |
          ku-pa-TGSReq-authenticator }
    }
}


AP-REQ-EXT      ::= [APPLICATION 18] Signed {
    [APPLICATION 18] SEQUENCE {
        pvno                [0] INTEGER (5),
        msg-type            [1] INTEGER (18),
        ap-options          [2] APOptions,
        ticket              [3] Ticket,
        authenticator       [4] EncryptedData {
            AuthenticatorExt,
            { key-session },
            { ku-APReq-authenticator |
              ku-pa-TGSReq-authenticator }
        },
        ...,
        extensions          [5] ApReqExtensions OPTIONAL,
        lang-tag            [6] SEQUENCE (SIZE (1..MAX))
                                OF LangTag OPTIONAL,
        ...
    }, { key-session }, { ku-APReq-cksum }
}


ApReqExtType    ::= TH-id

ApReqExtensions ::= SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    apReqExt-Type       [0] ApReqExtType,
    apReqExt-Data       [1] OCTET STRING
}


APOptions       ::= KerberosFlags { APOptionsBits }

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

```
    }
```

```
      -- plaintext of authenticator
      Authenticator1510       ::= [APPLICATION 2] SEQUENCE  {
          authenticator-vno   [0] INTEGER (5),
          crealm              [1] RealmIA5,
          cname               [2] PrincipalNameIA5,
          cksum               [3] Checksum {{ key-session },
              { ku-Authenticator-cksum |
                ku-pa-TGSReq-cksum }} OPTIONAL,
          cusec               [4] Microseconds,
          ctime               [5] KerberosTime,
          subkey              [6] EncryptionKey OPTIONAL,
          seq-number          [7] SeqNum32 OPTIONAL,
          authorization-data  [8] AuthorizationData OPTIONAL
      }

      AuthenticatorExt        ::= [APPLICATION 35] SEQUENCE  {
          authenticator-vno   [0] INTEGER (5),
          crealm              [1] RealmExt,
          cname               [2] PrincipalNameExt,
          cksum               [3] Checksum {{ key-session },
              { ku-Authenticator-cksum |
                ku-pa-TGSReq-cksum }} OPTIONAL,
          cusec               [4] Microseconds,
          ctime               [5] KerberosTime,
          subkey              [6] EncryptionKey OPTIONAL,
          seq-number          [7] SeqNum OPTIONAL,
          authorization-data  [8] AuthorizationData OPTIONAL,
          ...
      }


      AP-REP          ::= CHOICE {
          rfc1510     AP-REP-1510,
          ext         AP-REP-EXT
      }


      AP-REP-1510     ::= [APPLICATION 15] SEQUENCE {
          pvno        [0] INTEGER (5),
          msg-type    [1] INTEGER (15),
          enc-part    [2] EncryptedData {
              EncApRepPart1510,
              { key-session | key-subsession }, { ku-EncAPRepPart }}
      }
```

```
AP-REP-EXT      ::= [APPLICATION 19] Signed {
    [APPLICATION 19] SEQUENCE {
        pvno        [0] INTEGER (5),
        msg-type    [1] INTEGER (19),
        enc-part    [2] EncryptedData {
            EncAPRepPartExt,
            { key-session | key-subsession }, { ku-EncAPRepPart }},
        ...,
        extensions          [3] ApRepExtensions OPTIONAL,
        ...
    }, { key-session | key-subsession }, { ku-APRep-cksum }
}


ApRepExtType    ::= TH-id

ApRepExtensions ::= SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    apRepExt-Type       [0] ApRepExtType,
    apRepExt-Data       [1] OCTET STRING
}


EncAPRepPart    ::= CHOICE {
    rfc1510     EncAPRepPart1510,
    ext         EncAPRepPartExt
}


EncAPRepPart1510        ::= [APPLICATION 27] SEQUENCE {
    ctime               [0] KerberosTime,
    cusec               [1] Microseconds,
    subkey              [2] EncryptionKey OPTIONAL,
    seq-number          [3] SeqNum32 OPTIONAL
}


EncAPRepPartExt         ::= [APPLICATION 31] SEQUENCE {
    ctime               [0] KerberosTime,
    cusec               [1] Microseconds,
    subkey              [2] EncryptionKey OPTIONAL,
    seq-number          [3] SeqNum OPTIONAL,
    ...,
    authorization-data  [4] AuthorizationData OPTIONAL,
    ...
}


--
-- *** Application messages
--
```

```
KRB-SAFE        ::= CHOICE {
    rfc1510     KRB-SAFE-1510,
    ext         KRB-SAFE-EXT
}


KRB-SAFE-1510   ::= [APPLICATION 20] SEQUENCE {
    pvno        [0] INTEGER (5),
    msg-type    [1] INTEGER (20),
    safe-body   [2] KRB-SAFE-BODY,
    cksum       [3] ChecksumOf {
        KRB-SAFE-BODY,
        { key-session | key-subsession }, { ku-KrbSafe-cksum }}
}


-- Has safe-body optional to allow for GSS-MIC type functionality
KRB-SAFE-EXT    ::= [APPLICATION 34] SEQUENCE {
    pvno        [0] INTEGER (5),
    msg-type    [1] INTEGER (20),
    safe-body   [2] KRB-SAFE-BODY OPTIONAL,
    cksum       [3] ChecksumOf {
        KRB-SAFE-BODY,
        { key-session | key-subsession }, { ku-KrbSafe-cksum }},
    ...
}


KRB-SAFE-BODY   ::= SEQUENCE {
    user-data   [0] OCTET STRING,
    timestamp   [1] KerberosTime OPTIONAL,
    usec        [2] Microseconds OPTIONAL,
    seq-number  [3] SeqNum OPTIONAL,
    s-address   [4] HostAddress,
    r-address   [5] HostAddress OPTIONAL,
    ...         -- ASN.1 extensions must be excluded
                -- when sending to rfc1510 implementations
}


KRB-PRIV        ::= [APPLICATION 21] SEQUENCE {
    pvno        [0] INTEGER (5),
    msg-type    [1] INTEGER (21),
    enc-part    [3] EncryptedData {
        EncKrbPrivPart,
        { key-session | key-subsession }, { ku-EncKrbPrivPart }},
    ...
}
```

```
      EncKrbPrivPart  ::= [APPLICATION 28] SEQUENCE {
          user-data   [0] OCTET STRING,
          timestamp   [1] KerberosTime OPTIONAL,
          usec        [2] Microseconds OPTIONAL,
          seq-number  [3] SeqNum OPTIONAL,
          s-address   [4] HostAddress -- sender's addr --,
          r-address   [5] HostAddress OPTIONAL -- recip's addr --,
          ...         -- ASN.1 extensions must be excluded
                      -- when sending to rfc1510 implementations
      }


      KRB-CRED        ::= CHOICE {
          rfc1510     KRB-CRED-1510,
          ext         KRB-CRED-EXT

      }


      KRB-CRED-1510 ::= [APPLICATION 22] SEQUENCE {
          pvno        [0] INTEGER (5),
          msg-type    [1] INTEGER (22),
          tickets     [2] SEQUENCE OF Ticket,
          enc-part    [3] EncryptedData {
              EncKrbCredPart,
              { key-session | key-subsession }, { ku-EncKrbCredPart }}
      }


      KRB-CRED-EXT    ::= [APPLICATION 24] Signed {
          [APPLICATION 24] SEQUENCE {
              pvno        [0] INTEGER (5),
              msg-type    [1] INTEGER (24),
              tickets     [2] SEQUENCE OF Ticket,
              enc-part    [3] EncryptedData {
                  EncKrbCredPart,
                  { key-session | key-subsession }, { ku-EncKrbCredPart }},
              ...
          }, { key-session | key-subsession }, { ku-KrbCred-cksum }
      }



      EncKrbCredPart  ::= [APPLICATION 29] SEQUENCE {
          ticket-info [0] SEQUENCE OF KrbCredInfo,
          nonce       [1] Nonce OPTIONAL,
          timestamp   [2] KerberosTime OPTIONAL,
          usec        [3] Microseconds OPTIONAL,
          s-address   [4] HostAddress OPTIONAL,
          r-address   [5] HostAddress OPTIONAL
```

```
      }
```

```
KrbCredInfo     ::= SEQUENCE {
    key         [0] EncryptionKey,
    prealm      [1] Realm OPTIONAL,
    pname       [2] PrincipalName OPTIONAL,
    flags       [3] TicketFlags OPTIONAL,
    authtime    [4] KerberosTime OPTIONAL,
    starttime   [5] KerberosTime OPTIONAL,
    endtime     [6] KerberosTime OPTIONAL,
    renew-till  [7] KerberosTime OPTIONAL,
    srealm      [8] Realm OPTIONAL,
    sname       [9] PrincipalName OPTIONAL,
    caddr       [10] HostAddresses OPTIONAL
}


TGT-REQ         ::= [APPLICATION 16] SEQUENCE {
    pvno            [0] INTEGER (5),
    msg-type        [1] INTEGER (16),
    sname           [2] PrincipalName OPTIONAL,
    srealm          [3] Realm OPTIONAL,
    ...
}


--
-- *** Error messages
--


ErrCode ::= Int32

KRB-ERROR       ::= CHOICE {
    rfc1510     KRB-ERROR-1510,
    ext         KRB-ERROR-EXT
}
```

```
KRB-ERROR-1510 ::= [APPLICATION 30] SEQUENCE {
    pvno        [0] INTEGER (5),
    msg-type    [1] INTEGER (30),
    ctime       [2] KerberosTime OPTIONAL,
    cusec       [3] Microseconds OPTIONAL,
    stime       [4] KerberosTime,
    susec       [5] Microseconds,
    error-code  [6] ErrCode,
    crealm      [7] RealmIA5 OPTIONAL,
    cname       [8] PrincipalNameIA5 OPTIONAL,
    realm       [9] RealmIA5 -- Correct realm --,
    sname       [10] PrincipalNameIA5 -- Correct name --,
    e-text      [11] KerberosString OPTIONAL,
    e-data      [12] OCTET STRING OPTIONAL
}


KRB-ERROR-EXT ::= [APPLICATION 23] Signed {
    [APPLICATION 23] SEQUENCE{
        pvno        [0] INTEGER (5),
        msg-type    [1] INTEGER (23),
        ctime       [2] KerberosTime OPTIONAL,
        cusec       [3] Microseconds OPTIONAL,
        stime       [4] KerberosTime,
        susec       [5] Microseconds,
        error-code  [6] ErrCode,
        crealm      [7] Realm OPTIONAL,
        cname       [8] PrincipalName OPTIONAL,
        realm       [9] Realm -- Correct realm --,
        sname       [10] PrincipalName -- Correct name --,
        e-text      [11] KerberosString OPTIONAL,
        e-data      [12] OCTET STRING OPTIONAL,
        ...,
        typed-data          [13] TYPED-DATA OPTIONAL,
        nonce               [14] Nonce OPTIONAL,
        lang-tag            [15] LangTag OPTIONAL,
        ...
    }, { }, { ku-KrbError-cksum }
}


METHOD-DATA     ::= SEQUENCE OF PA-DATA


TDType ::= TH-id

TYPED-DATA      ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
    data-type   [0] TDType,
    data-value  [1] OCTET STRING OPTIONAL
```

```
    }
```

```
td-dh-parameters        TDType ::= int32 : 109


--
-- *** Error codes
--

-- No error
kdc-err-none                          ErrCode ::= 0
-- Client's entry in database has expired
kdc-err-name-exp                      ErrCode ::= 1
-- Server's entry in database has expired
kdc-err-service-exp                   ErrCode ::= 2
-- Requested protocol version number not supported
kdc-err-bad-pvno                      ErrCode ::= 3
-- Client's key encrypted in old master key
kdc-err-c-old-mast-kvno               ErrCode ::= 4
-- Server's key encrypted in old master key
kdc-err-s-old-mast-kvno               ErrCode ::= 5
-- Client not found in Kerberos database
kdc-err-c-principal-unknown           ErrCode ::= 6
-- Server not found in Kerberos database
kdc-err-s-principal-unknown           ErrCode ::= 7
-- Multiple principal entries in database
kdc-err-principal-not-unique          ErrCode ::= 8
-- The client or server has a null key
kdc-err-null-key                      ErrCode ::= 9
-- Ticket not eligible for postdating
kdc-err-cannot-postdate               ErrCode ::= 10
-- Requested start time is later than end time
kdc-err-never-valid                   ErrCode ::= 11
-- KDC policy rejects request
kdc-err-policy                        ErrCode ::= 12
-- KDC cannot accommodate requested option
kdc-err-badoption                     ErrCode ::= 13
-- KDC has no support for encryption type
kdc-err-etype-nosupp                  ErrCode ::= 14
-- KDC has no support for checksum type
kdc-err-sumtype-nosupp                ErrCode ::= 15
-- KDC has no support for padata type
kdc-err-padata-type-nosupp            ErrCode ::= 16
-- KDC has no support for transited type
kdc-err-trtype-nosupp                 ErrCode ::= 17
-- Clients credentials have been revoked
kdc-err-client-revoked                ErrCode ::= 18
-- Credentials for server have been revoked
kdc-err-service-revoked               ErrCode ::= 19
-- TGT has been revoked
kdc-err-tgt-revoked                   ErrCode ::= 20
```

```
      -- Client not yet valid - try again later
      kdc-err-client-notyet                ErrCode ::= 21
      -- Server not yet valid - try again later
      kdc-err-service-notyet               ErrCode ::= 22
      -- Password has expired - change password to reset
      kdc-err-key-expired                  ErrCode ::= 23
      -- Pre-authentication information was invalid
      kdc-err-preauth-failed               ErrCode ::= 24
      -- Additional pre-authenticationrequired
      kdc-err-preauth-required             ErrCode ::= 25
      -- Requested server and ticket don't match
      kdc-err-server-nomatch               ErrCode ::= 26
      -- Server principal valid for user2user only
      kdc-err-must-use-user2user           ErrCode ::= 27
      -- KDC Policy rejects transited path
      kdc-err-path-not-accpeted            ErrCode ::= 28
      -- A service is not available
      kdc-err-svc-unavailable              ErrCode ::= 29
      -- Integrity check on decrypted field failed
      krb-ap-err-bad-integrity             ErrCode ::= 31
      -- Ticket expired
      krb-ap-err-tkt-expired               ErrCode ::= 32
      -- Ticket not yet valid
      krb-ap-err-tkt-nyv                   ErrCode ::= 33
      -- Request is a replay
      krb-ap-err-repeat                    ErrCode ::= 34
      -- The ticket isn't for us
      krb-ap-err-not-us                    ErrCode ::= 35
      -- Ticket and authenticator don't match
      krb-ap-err-badmatch                  ErrCode ::= 36
      -- Clock skew too great
      krb-ap-err-skew                      ErrCode ::= 37
      -- Incorrect net address
      krb-ap-err-badaddr                   ErrCode ::= 38
      -- Protocol version mismatch
      krb-ap-err-badversion                ErrCode ::= 39
      -- Invalid msg type
      krb-ap-err-msg-type                  ErrCode ::= 40
```

```
      -- Message stream modified
      krb-ap-err-modified                  ErrCode ::= 41
      -- Message out of order
      krb-ap-err-badorder                  ErrCode ::= 42
      -- Specified version of key is not available
      krb-ap-err-badkeyver                 ErrCode ::= 44
      -- Service key not available
      krb-ap-err-nokey                     ErrCode ::= 45
      -- Mutual authentication failed
      krb-ap-err-mut-fail                  ErrCode ::= 46
      -- Incorrect message direction
      krb-ap-err-baddirection              ErrCode ::= 47
      -- Alternative authentication method required
      krb-ap-err-method                    ErrCode ::= 48
      -- Incorrect sequence number in message
      krb-ap-err-badseq                    ErrCode ::= 49
      -- Inappropriate type of checksum in message
      krb-ap-err-inapp-cksum               ErrCode ::= 50
      -- Policy rejects transited path
      krb-ap-path-not-accepted             ErrCode ::= 51
      -- Response too big for UDP, retry with TCP
      krb-err-response-too-big             ErrCode ::= 52
      -- Generic error (description in e-text)
      krb-err-generic                      ErrCode ::= 60
```

```
      -- Field is too long for this implementation
      krb-err-field-toolong                 ErrCode ::= 61
      -- Reserved for PKINIT
      kdc-error-client-not-trusted          ErrCode ::= 62
      -- Reserved for PKINIT
      kdc-error-kdc-not-trusted             ErrCode ::= 63
      -- Reserved for PKINIT
      kdc-error-invalid-sig                 ErrCode ::= 64
      -- Reserved for PKINIT
      kdc-err-key-too-weak                  ErrCode ::= 65
      -- Reserved for PKINIT
      kdc-err-certificate-mismatch          ErrCode ::= 66
      -- No TGT available to validate USER-TO-USER
      krb-ap-err-no-tgt                     ErrCode ::= 67
      -- USER-TO-USER TGT issued different KDC
      kdc-err-wrong-realm                   ErrCode ::= 68
      -- Ticket must be for USER-TO-USER
      krb-ap-err-user-to-user-required      ErrCode ::= 69
      -- Reserved for PKINIT
      kdc-err-cant-verify-certificate       ErrCode ::= 70
      -- Reserved for PKINIT
      kdc-err-invalid-certificate           ErrCode ::= 71
      -- Reserved for PKINIT
      kdc-err-revoked-certificate           ErrCode ::= 72
      -- Reserved for PKINIT
      kdc-err-revocation-status-unknown     ErrCode ::= 73
      -- Reserved for PKINIT
      kdc-err-revocation-status-unavailable ErrCode ::= 74


      END
```

B. Kerberos and Character Encodings (Informative)

[adapted from KCLAR 5.2.1]

The original specification of the Kerberos protocol in RFC 1510 uses GeneralString in numerous places for human-readable string data. Historical implementations of Kerberos cannot utilize the full power of GeneralString. This ASN.1 type requires the use of designation and invocation escape sequences as specified in ISO 2022 | ECMA-35 [ISO2022] to switch character sets, and the default character set that is designated as G0 is the ISO 646 | ECMA-6 [ISO646] International Reference Version (IRV) (aka U.S. ASCII), which mostly works.

ISO 2022 | ECMA-35 defines four character-set code elements (G0..G3) and two Control-function code elements (C0..C1). DER previously

[X690-1997] prohibited the designation of character sets as any but the G0 and C0 sets. This had the side effect of prohibiting the use

of (ISO Latin) character-sets such as ISO 8859-1 [ISO8859-1] or any other character-sets that utilize a 96-character set, since it is prohibited by ISO 2022 | ECMA-35 to designate them as the G0 code element. Recent revisions to the ASN.1 standards resolve this contradiction.

In practice, many implementations treat RFC 1510 GeneralStrings as if they were 8-bit strings of whichever character set the implementation defaults to, without regard for correct usage of character-set designation escape sequences. The default character set is often determined by the current user's operating system dependent locale. At least one major implementation places unescaped UTF-8 encoded Unicode characters in the GeneralString. This failure to conform to the GeneralString specifications results in interoperability issues when conflicting character encodings are utilized by the Kerberos clients, services, and KDC.

This unfortunate situation is the result of improper documentation of the restrictions of the ASN.1 GeneralString type in prior Kerberos specifications.

[the following should probably be rewritten and moved into the principal name section]

For compatibility, implementations MAY choose to accept GeneralString values that contain characters other than those permitted by IA5String, but they should be aware that character set designation codes will likely be absent, and that the encoding should probably be treated as locale-specific in almost every way. Implementations MAY also choose to emit GeneralString values that are beyond those permitted by IA5String, but should be aware that doing so is extraordinarily risky from an interoperability perspective.

Some existing implementations use GeneralString to encode unescaped locale-specific characters. This is a violation of the ASN.1 standard. Most of these implementations encode US-ASCII in the left-hand half, so as long the implementation transmits only US-ASCII, the ASN.1 standard is not violated in this regard. As soon as such an implementation encodes unescaped locale-specific characters with the high bit set, it violates the ASN.1 standard.

Other implementations have been known to use GeneralString to contain a UTF-8 encoding. This also violates the ASN.1 standard, since UTF-8 is a different encoding, not a 94 or 96 character "G" set as defined by ISO 2022. It is believed that these implementations do not even use the ISO 2022 escape sequence to change the character encoding. Even if implementations were to announce the change of encoding by using that escape sequence, the ASN.1 standard prohibits the use of any escape sequences other than those used to designate/invoke "G" or

"C" sets allowed by GeneralString.

C. Kerberos History (Informative)

[Adapted from KCLAR "BACKGROUND"]

The Kerberos model is based in part on Needham and Schroeder's trusted third-party authentication protocol [NS78] and on modifications suggested by Denning and Sacco [DS81]. The original design and implementation of Kerberos Versions 1 through 4 was the work of two former Project Athena staff members, Steve Miller of Digital Equipment Corporation and Clifford Neuman (now at the Information Sciences Institute of the University of Southern California), along with Jerome Saltzer, Technical Director of Project Athena, and Jeffrey Schiller, MIT Campus Network Manager. Many other members of Project Athena have also contributed to the work on Kerberos.

Version 5 of the Kerberos protocol (described in this document) has evolved from Version 4 based on new requirements and desires for features not available in Version 4. The design of Version 5 of the Kerberos protocol was led by Clifford Neuman and John Kohl with much input from the community. The development of the MIT reference implementation was led at MIT by John Kohl and Theodore Ts'o, with help and contributed code from many others. Since RFC1510 was issued, extensions and revisions to the protocol have been proposed by many individuals. Some of these proposals are reflected in this document. Where such changes involved significant effort, the document cites the contribution of the proposer.

Reference implementations of both version 4 and version 5 of Kerberos are publicly available and commercial implementations have been developed and are widely used. Details on the differences between Kerberos Versions 4 and 5 can be found in [KNT94].

D. Notational Differences from [KCLAR]

[possible point for discussion]

[KCLAR] uses notational conventions slightly different from this document. As a derivative of RFC 1510, the text of [KCLAR] uses C-language style identifier names for defined values. In ASN.1 notation, identifiers referencing defined values must begin with a lowercase letter and contain hyphen (-) characters rather than underscore (_) characters, while identifiers referencing types begin with an uppercase letter. [KCLAR] and RFC 1510 use all-uppercase identifiers with underscores to identify defined values. This has the potential to create confusion, but neither document defines values using actual ASN.1 value-assignment notation.

It is debatable whether it is advantageous to write all identifier

names (regardless of their ASN.1 token type) in all-uppercase letters for the purpose of emphasis in running text. The alternative is to

use double-quote characters (") when ambiguity is possible.

Normative References

[ISO646]
"7-bit coded character set", ISO/IEC 646:1991 | ECMA-6:1991.

[ISO2022]
"Information technology -- Character code structure and extension techniques", ISO/IEC 2022:1994 | ECMA-35:1994.

[KCRYPTO]
K. Raeburn, "Encryption and Checksum Specifications for Kerberos 5", draft-ietf-krb-wg-crypto-07.txt, work in progress.

[RFC2119]
S. Bradner, RFC2119: "Key words for use in RFC's to Indicate Requirement Levels", March 1997.

[RFC3660]
H. Alvestrand, "Tags for the Identification of Languages", RFC 3660, January 2001.

[SASLPREP]
Kurt D. Zeilenga, "SASLprep: Stringprep profile for user names and passwords", draft-ietf-sasl-saslprep-10.txt, work in progress.

[X680]
"Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002.

[X682]
"Information technology -- Abstract Syntax Notation One (ASN.1): Constraint specification", ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3:2002.

[X683]
"Information technology -- Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications", ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002.

[X690]
"Information technology -- ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002.

Informative References

[DS81]
Dorothy E. Denning and Giovanni Maria Sacco, "Time-stamps in Key Distribution Protocols," Communications of the ACM, Vol. 24(8), pp. 533-536 (August 1981).

[Dub00]
Olivier Dubuisson, "ASN.1 - Communication between Heterogeneous Systems", Elsevier-Morgan Kaufmann, 2000. <http://www.oss.com/asn1/dubuisson.html>

[ISO8859-1]
"Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO/IEC 8859-1:1998.

[KCLAR]
Clifford Neuman, Tom Yu, Sam Hartman, Ken Raeburn, "The Kerberos Network Authentication Service (V5)", draft-ietf-krb-wg-kerberos-clarifications-07.txt, work in progress.

[KNT94]
John T. Kohl, B. Clifford Neuman, and Theodore Y. Ts'o, "The Evolution of the Kerberos Authentication System". In Distributed Open Systems, pages 78-94. IEEE Computer Society Press, 1994.

[Lar96]
John Larmouth, "Understanding OSI", International Thomson Computer Press, 1996. <http://www.isi.salford.ac.uk/books/osi.html>

[Lar99]
John Larmouth, "ASN.1 Complete", Elsevier-Morgan Kaufmann, 1999. <http://www.oss.com/asn1/larmouth.html>

[NS78]
Roger M. Needham and Michael D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers", Communications of the ACM, Vol. 21(12), pp. 993-999 (December, 1978).

[RFC1510]
J. Kohl and B. C. Neuman, "The Kerberos Network Authentication Service (v5)", RFC1510, September 1993, Status: Proposed Standard.

[RFC1964]
J. Linn, "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996, Status: Proposed Standard.

[X690-2002]
"Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002.

Author's Address

Tom Yu
77 Massachusetts Ave
Cambridge, MA 02139
USA
tlyu@mit.edu

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-

ipr@ietf.org.