Network Working Group
Request for Comments: 4120
Obsoletes: 1510
Category: Standards Track

C. Neuman
USC-ISI
T. Yu
S. Hartman
K. Raeburn
MIT

July 2005

The Kerberos Network Authentication Service (V5)

Status of This Memo

Copyright Notice

Abstract

This document provides an overview and specification of Version 5 of the Kerberos protocol, and it obsoletes RFC 1510 to clarify aspects of the protocol and its intended use that require more detailed or clearer explanation than was provided in RFC 1510. This document is intended to provide a detailed description of the protocol, suitable for implementation, together with descriptions of the appropriate use of protocol messages and fields within those messages.

Table of Contents

1. Introduction

This document describes the concepts and model upon which the Kerberos network authentication system is based. It also specifies Version 5 of the Kerberos protocol. The motivations, goals, assumptions, and rationale behind most design decisions are treated cursorily; they are more fully described in a paper available in IEEE communications [NT94] and earlier in the Kerberos portion of the Athena Technical Plan [MNSS87].

This document is not intended to describe Kerberos to the end user, system administrator, or application developer. Higher-level papers describing Version 5 of the Kerberos system [NT94] and documenting version 4 [SNS88] are available elsewhere.

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 because of new requirements and desires for features not available in Version 4. The design of Version 5 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 RFC 1510 was issued, many individuals have proposed extensions and revisions to the protocol. This document reflects some of these proposals. 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 Versions 4 and 5 can be found in [KNT94].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.1. The Kerberos Protocol

Kerberos provides a means of verifying the identities of principals, (e.g., a workstation user or a network server) on an open (unprotected) network. This is accomplished without relying on assertions by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will. Kerberos performs authentication under these conditions as a trusted third-party authentication service by using conventional (shared secret key) cryptography. Extensions to Kerberos (outside the scope of this document) can provide for the use of public key cryptography during certain phases of the authentication protocol. Such extensions support Kerberos authentication for users registered with public key certification authorities and provide certain benefits of public key cryptography in situations where they are needed.

The basic Kerberos authentication process proceeds as follows: A client sends a request to the authentication server (AS) for "credentials" for a given server. The AS responds with these credentials, encrypted in the client's key. The credentials consist of a "ticket" for the server and a temporary encryption key (often called a "session key"). The client transmits the ticket (which contains the client's identity and a copy of the session key, all encrypted in the server's key) to the server. The session key (now shared by the client and server) is used to authenticate the client and may optionally be used to authenticate the server. It may also be used to encrypt further communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication. Note that many applications use Kerberos' functions only upon the initiation of a stream-based network connection. Unless an application performs encryption or integrity protection for the data stream, the identity verification applies only to the initiation of the connection, and it does not guarantee that subsequent messages on the connection originate from the same principal.

Implementation of the basic protocol consists of one or more authentication servers running on physically secure hosts. The authentication servers maintain a database of principals (i.e., users and servers) and their secret keys. Code libraries provide encryption and implement the Kerberos protocol. In order to add authentication to its transactions, a typical network application adds calls to the Kerberos library directly or through the Generic Security Services Application Programming Interface (GSS-API) described in a separate document [RFC4121]. These calls result in the transmission of the messages necessary to achieve authentication.

The Kerberos protocol consists of several sub-protocols (or exchanges). There are two basic methods by which a client can ask a Kerberos server for credentials. In the first approach, the client sends a cleartext request for a ticket for the desired server to the AS. The reply is sent encrypted in the client's secret key. Usually this request is for a ticket-granting ticket (TGT), which can later be used with the ticket-granting server (TGS). In the second method, the client sends a request to the TGS. The client uses the TGT to authenticate itself to the TGS in the same manner as if it were contacting any other application server that requires Kerberos authentication. The reply is encrypted in the session key from the TGT. Though the protocol specification describes the AS and the TGS as separate servers, in practice they are implemented as different protocol entry points within a single Kerberos server.

Once obtained, credentials may be used to verify the identity of the principals in a transaction, to ensure the integrity of messages exchanged between them, or to preserve privacy of the messages. The application is free to choose whatever protection may be necessary.

To verify the identities of the principals in a transaction, the client transmits the ticket to the application server. Because the ticket is sent "in the clear" (parts of it are encrypted, but this encryption doesn't thwart replay) and might be intercepted and reused by an attacker, additional information is sent to prove that the message originated with the principal to whom the ticket was issued. This information (called the authenticator) is encrypted in the session key and includes a timestamp. The timestamp proves that the message was recently generated and is not a replay. Encrypting the authenticator in the session key proves that it was generated by a party possessing the session key. Since no one except the requesting principal and the server know the session key (it is never sent over the network in the clear), this guarantees the identity of the client.

The integrity of the messages exchanged between principals can also be guaranteed by using the session key (passed in the ticket and contained in the credentials). This approach provides detection of both replay attacks and message stream modification attacks. It is accomplished by generating and transmitting a collision-proof checksum (elsewhere called a hash or digest function) of the client's message, keyed with the session key. Privacy and integrity of the messages exchanged between principals can be secured by encrypting the data to be passed by using the session key contained in the ticket or the sub-session key found in the authenticator.

The authentication exchanges mentioned above require read-only access to the Kerberos database. Sometimes, however, the entries in the database must be modified, such as when adding new principals or changing a principal's key. This is done using a protocol between a client and a third Kerberos server, the Kerberos Administration Server (KADM). There is also a protocol for maintaining multiple copies of the Kerberos database. Neither of these protocols are described in this document.

1.2. Cross-Realm Operation

The Kerberos protocol is designed to operate across organizational boundaries. A client in one organization can be authenticated to a server in another. Each organization wishing to run a Kerberos server establishes its own "realm". The name of the realm in which a client is registered is part of the client's name and can be used by the end-service to decide whether to honor a request.

By establishing "inter-realm" keys, the administrators of two realms can allow a client authenticated in the local realm to prove its identity to servers in other realms. The exchange of inter-realm keys (a separate key may be used for each direction) registers the ticket-granting service of each realm as a principal in the other realm. A client is then able to obtain a TGT for the remote realm's ticket-granting service from its local realm. When that TGT is used, the remote ticket-granting service uses the inter-realm key (which usually differs from its own normal TGS key) to decrypt the TGT; thus it is certain that the ticket was issued by the client's own TGS. Tickets issued by the remote ticket-granting service will indicate to the end-service that the client was authenticated from another realm.

Without cross-realm operation, and with appropriate permission, the client can arrange registration of a separately-named principal in a remote realm and engage in normal exchanges with that realm's services. However, for even small numbers of clients this becomes cumbersome, and more automatic methods as described here are necessary.

A realm is said to communicate with another realm if the two realms share an inter-realm key, or if the local realm shares an inter-realm key with an intermediate realm that communicates with the remote realm. An authentication path is the sequence of intermediate realms that are transited in communicating from one realm to another.

Realms may be organized hierarchically. Each realm shares a key with its parent and a different key with each child. If an inter-realm key is not directly shared by two realms, the hierarchical organization allows an authentication path to be easily constructed.

If a hierarchical organization is not used, it may be necessary to consult a database in order to construct an authentication path between realms.

Although realms are typically hierarchical, intermediate realms may be bypassed to achieve cross-realm authentication through alternate authentication paths. (These might be established to make communication between two realms more efficient.) It is important for the end-service to know which realms were transited when deciding how much faith to place in the authentication process. To facilitate this decision, a field in each ticket contains the names of the realms that were involved in authenticating the client.

The application server is ultimately responsible for accepting or rejecting authentication and SHOULD check the transited field. The application server may choose to rely on the Key Distribution Center (KDC) for the application server's realm to check the transited field. The application server's KDC will set the TRANSITED-POLICY-CHECKED flag in this case. The KDCs for intermediate realms may also check the transited field as they issue TGTs for other realms, but they are encouraged not to do so. A client may request that the KDCs not check the transited field by setting the DISABLE-TRANSITED-CHECK flag. KDCs SHOULD honor this flag.

1.3. Choosing a Principal with Which to Communicate

The Kerberos protocol provides the means for verifying (subject to the assumptions in Section 1.6) that the entity with which one communicates is the same entity that was registered with the KDC using the claimed identity (principal name). It is still necessary to determine whether that identity corresponds to the entity with which one intends to communicate.

When appropriate data has been exchanged in advance, the application may perform this determination syntactically based on the application protocol specification, information provided by the user, and configuration files. For example, the server principal name (including realm) for a telnet server might be derived from the user-specified host name (from the telnet command line), the "host/" prefix specified in the application protocol specification, and a mapping to a Kerberos realm derived syntactically from the domain part of the specified hostname and information from the local Kerberos realms database.

One can also rely on trusted third parties to make this determination, but only when the data obtained from the third party is suitably integrity-protected while resident on the third-party

server and when transmitted. Thus, for example, one should not rely on an unprotected DNS record to map a host alias to the primary name of a server, accepting the primary name as the party that one intends to contact, since an attacker can modify the mapping and impersonate the party.

Implementations of Kerberos and protocols based on Kerberos MUST NOT use insecure DNS queries to canonicalize the hostname components of the service principal names (i.e., they MUST NOT use insecure DNS queries to map one name to another to determine the host part of the principal name with which one is to communicate). In an environment without secure name service, application authors MAY append a statically configured domain name to unqualified hostnames before passing the name to the security mechanisms, but they should do no more than that. Secure name service facilities, if available, might be trusted for hostname canonicalization, but such canonicalization by the client SHOULD NOT be required by KDC implementations.

Implementation note: Many current implementations do some degree of canonicalization of the provided service name, often using DNS even though it creates security problems. However, there is no consistency among implementations as to whether the service name is case folded to lowercase or whether reverse resolution is used. To maximize interoperability and security, applications SHOULD provide security mechanisms with names that result from folding the user-entered name to lowercase without performing any other modifications or canonicalization.

1.4. Authorization

As an authentication service, Kerberos provides a means of verifying the identity of principals on a network. Authentication is usually useful primarily as a first step in the process of authorization, determining whether a client may use a service, which objects the client is allowed to access, and the type of access allowed for each. Kerberos does not, by itself, provide authorization. Possession of a client ticket for a service provides only for authentication of the client to that service, and in the absence of a separate authorization procedure, an application should not consider it to authorize the use of that service.

Separate authorization methods MAY be implemented as application-specific access control functions and may utilize files on the application server, on separately issued authorization credentials such as those based on proxies [Neu93], or on other authorization services. Separately authenticated authorization credentials MAY be embedded in a ticket's authorization data when encapsulated by the KDC-issued authorization data element.

Applications should not accept the mere issuance of a service ticket by the Kerberos server (even by a modified Kerberos server) as granting authority to use the service, since such applications may become vulnerable to the bypass of this authorization check in an environment where other options for application authentication are provided, or if they interoperate with other KDCs.

1.5. Extending Kerberos without Breaking Interoperability

As the deployed base of Kerberos implementations grows, extending Kerberos becomes more important. Unfortunately, some extensions to the existing Kerberos protocol create interoperability issues because of uncertainty regarding the treatment of certain extensibility options by some implementations. This section includes guidelines that will enable future implementations to maintain interoperability.

Kerberos provides a general mechanism for protocol extensibility. Some protocol messages contain typed holes -- sub-messages that contain an octet-string along with an integer that defines how to interpret the octet-string. The integer types are registered centrally, but they can be used both for vendor extensions and for extensions standardized through the IETF.

In this document, the word "extension" refers to extension by defining a new type to insert into an existing typed hole in a protocol message. It does not refer to extension by addition of new fields to ASN.1 types, unless the text explicitly indicates otherwise.

1.5.1. Compatibility with RFC 1510

Note that existing Kerberos message formats cannot readily be extended by adding fields to the ASN.1 types. Sending additional fields often results in the entire message being discarded without an error indication. Future versions of this specification will provide guidelines to ensure that ASN.1 fields can be added without creating an interoperability problem.

In the meantime, all new or modified implementations of Kerberos that receive an unknown message extension SHOULD preserve the encoding of the extension but otherwise ignore its presence. Recipients MUST NOT decline a request simply because an extension is present.

There is one exception to this rule. If an unknown authorization data element type is received by a server other than the ticket-granting service either in an AP-REQ or in a ticket contained in an AP-REQ, then authentication MUST fail. One of the primary uses of authorization data is to restrict the use of the ticket. If the

service cannot determine whether the restriction applies to that service, then a security weakness may result if the ticket can be used for that service. Authorization elements that are optional SHOULD be enclosed in the AD-IF-RELEVANT element.

The ticket-granting service MUST ignore but propagate to derivative tickets any unknown authorization data types, unless those data types are embedded in a MANDATORY-FOR-KDC element, in which case the request will be rejected. This behavior is appropriate because requiring that the ticket-granting service understand unknown authorization data types would require that KDC software be upgraded to understand new application-level restrictions before applications used these restrictions, decreasing the utility of authorization data as a mechanism for restricting the use of tickets. No security problem is created because services to which the tickets are issued will verify the authorization data.

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

1.5.2. Sending Extensible Messages

Care must be taken to ensure that old implementations can understand messages sent to them, even if they do not understand an extension that is used. Unless the sender knows that an 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 implementations that do not interoperate is easier when errors are returned.

1.6. Environmental Assumptions

Kerberos imposes a few assumptions on the environment in which it can properly function, including the following:

* "Denial of service" attacks are not solved with Kerberos. There are places in the protocols where an intruder can prevent an application from participating in the proper authentication steps. Detection and solution of such attacks (some of which can appear to be not-uncommon "normal" failure modes for the system) are usually best left to the human administrators and users.

* Principals MUST keep their secret keys secret. If an intruder somehow steals a principal's key, it will be able to masquerade as that principal or to impersonate any server to the legitimate principal.

* "Password guessing" attacks are not solved by Kerberos. If a user chooses a poor password, it is possible for an attacker to successfully mount an offline dictionary attack by repeatedly attempting to decrypt, with successive entries from a dictionary, messages obtained which are encrypted under a key derived from the user's password.

* Each host on the network MUST have a clock which is "loosely synchronized" to the time of the other hosts; this synchronization is used to reduce the bookkeeping needs of application servers when they do replay detection. The degree of "looseness" can be configured on a per-server basis, but it is typically on the order of 5 minutes. If the clocks are synchronized over the network, the clock synchronization protocol MUST itself be secured from network attackers.

* Principal identifiers are not recycled on a short-term basis. A typical mode of access control will use access control lists (ACLs) to grant permissions to particular principals. If a stale ACL entry remains for a deleted principal and the principal identifier is reused, the new principal will inherit rights specified in the stale ACL entry. By not re-using principal identifiers, the danger of inadvertent access is removed.

1.7. Glossary of Terms

Below is a list of terms used throughout this document.

Authentication
: Verifying the claimed identity of a principal.

Authentication header
: A record containing a Ticket and an Authenticator to be presented to a server as part of the authentication process.

Authentication path
: A sequence of intermediate realms transited in the authentication process when communicating from one realm to another.

Authenticator
: A record containing information that can be shown to have been recently generated using the session key known only by the client and server.

Authorization
: The process of determining whether a client may use a service, which objects the client is allowed to access, and the type of access allowed for each.

Capability
: A token that grants the bearer permission to access an object or service. In Kerberos, this might be a ticket whose use is restricted by the contents of the authorization data field, but which lists no network addresses, together with the session key necessary to use the ticket.

Ciphertext
: The output of an encryption function. Encryption transforms plaintext into ciphertext.

Client
: A process that makes use of a network service on behalf of a user. Note that in some cases a Server may itself be a client of some other server (e.g., a print server may be a client of a file server).

Credentials
: A ticket plus the secret session key necessary to use that ticket successfully in an authentication exchange.

Encryption Type (etype)
: When associated with encrypted data, an encryption type identifies the algorithm used to encrypt the data and is used to select the appropriate algorithm for decrypting the data. Encryption type tags are communicated in other messages to enumerate algorithms that are desired, supported, preferred, or allowed to be used for encryption of data between parties. This preference is combined with local information and policy to select an algorithm to be used.

KDC
: Key Distribution Center. A network service that supplies tickets and temporary session keys; or an instance of that service or the

host on which it runs. The KDC services both initial ticket and ticket-granting ticket requests. The initial ticket portion is sometimes referred to as the Authentication Server (or service). The ticket-granting ticket portion is sometimes referred to as the ticket-granting server (or service).

Kerberos
: The name given to the Project Athena's authentication service, the protocol used by that service, or the code used to implement the authentication service. The name is adopted from the three-headed dog that guards Hades.

Key Version Number (kvno)
: A tag associated with encrypted data identifies which key was used for encryption when a long-lived key associated with a principal changes over time. It is used during the transition to a new key so that the party decrypting a message can tell whether the data was encrypted with the old or the new key.

Plaintext
: The input to an encryption function or the output of a decryption function. Decryption transforms ciphertext into plaintext.

Principal
: A named client or server entity that participates in a network communication, with one name that is considered canonical.

Principal identifier
: The canonical name used to identify each different principal uniquely.

Seal
: To encipher a record containing several fields in such a way that the fields cannot be individually replaced without knowledge of the encryption key or leaving evidence of tampering.

Secret key
: An encryption key shared by a principal and the KDC, distributed outside the bounds of the system, with a long lifetime. In the case of a human user's principal, the secret key MAY be derived from a password.

Server
: A particular Principal that provides a resource to network clients. The server is sometimes referred to as the Application Server.

Service
A resource provided to network clients; often provided by more than one server (for example, remote file service).

Session key
A temporary encryption key used between two principals, with a lifetime limited to the duration of a single login "session". In the Kerberos system, a session key is generated by the KDC. The session key is distinct from the sub-session key, described next.

Sub-session key
A temporary encryption key used between two principals, selected and exchanged by the principals using the session key, and with a lifetime limited to the duration of a single association. The sub-session key is also referred to as the subkey.

Ticket
A record that helps a client authenticate itself to a server; it contains the client's identity, a session key, a timestamp, and other information, all sealed using the server's secret key. It only serves to authenticate a client when presented along with a fresh Authenticator.

2. Ticket Flag Uses and Requests

Each Kerberos ticket contains a set of flags that are used to indicate attributes of that ticket. Most flags may be requested by a client when the ticket is obtained; some are automatically turned on and off by a Kerberos server as required. The following sections explain what the various flags mean and give examples of reasons to use them. With the exception of the INVALID flag, clients MUST ignore ticket flags that are not recognized. KDCs MUST ignore KDC options that are not recognized. Some implementations of RFC 1510 are known to reject unknown KDC options, so clients may need to resend a request without new KDC options if the request was rejected when sent with options added since RFC 1510. Because new KDCs will ignore unknown options, clients MUST confirm that the ticket returned by the KDC meets their needs.

Note that it is not, in general, possible to determine whether an option was not honored because it was not understood or because it was rejected through either configuration or policy. When adding a new option to the Kerberos protocol, designers should consider whether the distinction is important for their option. If it is, a mechanism for the KDC to return an indication that the option was understood but rejected needs to be provided in the specification of the option. Often in such cases, the mechanism needs to be broad enough to permit an error or reason to be returned.

2.1. Initial, Pre-authenticated, and Hardware-Authenticated Tickets

The INITIAL flag indicates that a ticket was issued using the AS protocol, rather than issued based on a TGT. Application servers that want to require the demonstrated knowledge of a client's secret key (e.g., a password-changing program) can insist that this flag be set in any tickets they accept, and can thus be assured that the client's key was recently presented to the authentication server.

The PRE-AUTHENT and HW-AUTHENT flags provide additional information about the initial authentication, regardless of whether the current ticket was issued directly (in which case INITIAL will also be set) or issued on the basis of a TGT (in which case the INITIAL flag is clear, but the PRE-AUTHENT and HW-AUTHENT flags are carried forward from the TGT).

2.2. Invalid Tickets

The INVALID flag indicates that a ticket is invalid. Application servers MUST reject tickets that have this flag set. A postdated ticket will be issued in this form. Invalid tickets MUST be validated by the KDC before use, by being presented 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 that have been stolen before their starttime can be rendered permanently invalid (through a hot-list mechanism) (see Section 3.3.3.1).

2.3. Renewable Tickets

Applications may desire to hold tickets that can be valid for long periods of time. However, this can expose their 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 client to have long-term access to its secret key, an even greater risk. Renewable tickets can be used to mitigate the consequences of theft. 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 (i.e., before it expires) present a renewable ticket to the KDC, with the RENEW option set 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 whether the ticket had been reported stolen since its

last renewal; it will refuse to renew 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 (discussed below in Section 3.3). 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 reset by default, but a client MAY request it be set by setting the RENEWABLE option in the KRB_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.

2.4. Postdated Tickets

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 ticket-granting service. It can be ignored by application servers. This flag MUST be set in a TGT in order to issue a postdated ticket based on the presented ticket. It is reset by default; a client MAY request it by setting the ALLOW-POSTDATE option in the KRB_AS_REQ message. This flag does not allow a client to obtain a postdated TGT; postdated TGTs can only be obtained by requesting the postdating in the KRB_AS_REQ message. The life (endtime-starttime) of a postdated ticket will be the remaining life of the TGT at the time of the request, unless the RENEWABLE option is also set, in which case it can be the full life (endtime-starttime) of the TGT. 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 to be validated before use.

2.5. Proxiable and Proxy Tickets

At times 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 do this by granting it a proxy.

The process of granting a proxy by 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 TGT.

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 TGT) 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 TGT, and that it be 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 are often valid only from those network addresses specifically included in the ticket, but it is permissible as a policy option to allow requests and to issue tickets with no network addresses specified. 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.

2.6. Forwardable Tickets

Authentication forwarding is an instance of a proxy where the service that is granted is complete use of the client's identity. An example of 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 TGTs 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 TGT.

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, specifies 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.

2.7. Transited Policy Checking

In Kerberos, the application server is ultimately responsible for accepting or rejecting authentication, and it SHOULD check that only suitably trusted KDCs are relied upon to authenticate a principal. The transited field in the ticket identifies which realms (and thus which KDCs) were involved in the authentication process, and an application server would normally check this field. If any of these are untrusted to authenticate the indicated client principal (probably determined by a realm-based policy), the authentication attempt MUST be rejected. The presence of trusted KDCs in this list does not provide any guarantee; an untrusted KDC may have fabricated the list.

Although the end server ultimately decides whether authentication is valid, the KDC for the end server's realm MAY apply a realm-specific policy for validating the transited field and accepting credentials for cross-realm authentication. When the KDC applies such checks and accepts such cross-realm authentication, it will set the TRANSITED-POLICY-CHECKED flag in the service tickets it issues based

on the cross-realm TGT. A client MAY request that the KDCs not check the transited field by setting the DISABLE-TRANSITED-CHECK flag. KDCs are encouraged but not required to honor this flag.

Application servers MUST either do the transited-realm checks themselves or reject cross-realm tickets without TRANSITED-POLICY-CHECKED set.

2.8. OK as Delegate

For some applications, a client may need to delegate authority to a server to act on its behalf in contacting other services. This requires that the client forward credentials to an intermediate server. The ability for a client to obtain a service ticket to a server conveys no information to the client about whether the server should be trusted to accept delegated credentials. The OK-AS-DELEGATE provides a way for a KDC to communicate local realm policy to a client regarding whether an intermediate server is trusted to accept such credentials.

The copy of the ticket flags in the encrypted part of the KDC reply may have the OK-AS-DELEGATE flag set to indicate to the client that the server specified in the ticket has been determined by the policy of the realm to be a suitable recipient of delegation. A client can use the presence of this flag to help it decide whether to delegate credentials (grant either a proxy or a forwarded TGT) to this server. 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.

2.9. Other KDC Options

There are three additional options that MAY be set in a client's request of the KDC.

2.9.1. Renewable-OK

The RENEWABLE-OK option indicates that the client will accept a renewable ticket if a ticket with the requested life cannot otherwise be provided. If a ticket with the requested life cannot be provided, then the KDC MAY issue a renewable ticket with a renew-till equal to the requested endtime. The value of the renew-till field MAY still be adjusted by site-determined limits or limits imposed by the individual principal or server.

2.9.2. ENC-TKT-IN-SKEY

In its basic form, the Kerberos protocol supports authentication in a client-server setting and is not well suited to authentication in a peer-to-peer environment because the long-term key of the user does not remain on the workstation after initial login. Authentication of such peers may be supported by Kerberos in its user-to-user variant. The ENC-TKT-IN-SKEY option supports user-to-user authentication by allowing the KDC to issue a service ticket encrypted using the session key from another TGT issued to another user. The ENC-TKT-IN-SKEY option is honored only by the ticket-granting service. It indicates that the ticket to be issued for the end server is to be encrypted in the session key from the additional second TGT provided with the request. See Section 3.3.3 for specific details.

2.9.3. Passwordless Hardware Authentication

The OPT-HARDWARE-AUTH option indicates that the client wishes to use some form of hardware authentication instead of or in addition to the client's password or other long-lived encryption key. OPT-HARDWARE-AUTH is honored only by the authentication service. If supported and allowed by policy, the KDC will return an error code of KDC_ERR_PREAUTH_REQUIRED and include the required METHOD-DATA to perform such authentication.

3. Message Exchanges

The following sections describe the interactions between network clients and servers and the messages involved in those exchanges.

3.1. The Authentication Service Exchange

Summary

Message direction	Message type	Section
1. Client to Kerberos	KRB_AS_REQ	5.4.1
2. Kerberos to client	KRB_AS_REP or KRB_ERROR	5.4.2 5.9.1

The Authentication Service (AS) Exchange between the client and the Kerberos Authentication Server is initiated by a client when it wishes to obtain authentication credentials for a given server but currently holds no credentials. In its basic form, the client's secret key is used for encryption and decryption. This exchange is typically used at the initiation of a login session to obtain credentials for a Ticket-Granting Server, which will subsequently be used to obtain credentials for other servers (see Section 3.3)

without requiring further use of the client's secret key. This exchange is also used to request credentials for services that must not be mediated through the Ticket-Granting Service, but rather require knowledge of a principal's secret key, such as the password-changing service (the password-changing service denies requests unless the requester can demonstrate knowledge of the user's old password; requiring this knowledge prevents unauthorized password changes by someone walking up to an unattended session).

This exchange does not by itself provide any assurance of the identity of the user. To authenticate a user logging on to a local system, the credentials obtained in the AS exchange may first be used in a TGS exchange to obtain credentials for a local server; those credentials must then be verified by a local server through successful completion of the Client/Server exchange.

The AS exchange consists of two messages: KRB_AS_REQ from the client to Kerberos, and KRB_AS_REP or KRB_ERROR in reply. The formats for these messages are described in Sections 5.4.1, 5.4.2, and 5.9.1.

In the request, the client sends (in cleartext) its own identity and the identity of the server for which it is requesting credentials, other information about the credentials it is requesting, and a randomly generated nonce, which can be used to detect replays and to associate replies with the matching requests. This nonce MUST be generated randomly by the client and remembered for checking against the nonce in the expected reply. The response, KRB_AS_REP, contains a ticket for the client to present to the server, and a session key that will be shared by the client and the server. The session key and additional information are encrypted in the client's secret key. The encrypted part of the KRB_AS_REP message also contains the nonce that MUST be matched with the nonce from the KRB_AS_REQ message.

Without pre-authentication, the authentication server does not know whether the client is actually the principal named in the request. It simply sends a reply without knowing or caring whether they are the same. This is acceptable because nobody but the principal whose identity was given in the request will be able to use the reply. Its critical information is encrypted in that principal's key. However, an attacker can send a KRB_AS_REQ message to get known plaintext in order to attack the principal's key. Especially if the key is based on a password, this may create a security exposure. So the initial request supports an optional field that can be used to pass additional information that might be needed for the initial exchange. This field SHOULD be used for pre-authentication as described in sections 3.1.1 and 5.2.7.

Various errors can occur; these are indicated by an error response (KRB_ERROR) instead of the KRB_AS_REP response. The error message is not encrypted. The KRB_ERROR message contains information that can be used to associate it with the message to which it replies. The contents of the KRB_ERROR message are not integrity-protected. As such, the client cannot detect replays, fabrications, or modifications. A solution to this problem will be included in a future version of the protocol.

3.1.1. Generation of KRB_AS_REQ Message

The client may specify a number of options in the initial request. Among these options are whether pre-authentication is to be performed; whether the requested ticket is to be renewable, proxiable, or forwardable; whether it should be postdated or allow postdating of derivative tickets; and whether a renewable ticket will be accepted in lieu of a non-renewable ticket if the requested ticket expiration date cannot be satisfied by a non-renewable ticket (due to configuration constraints).

The client prepares the KRB_AS_REQ message and sends it to the KDC.

3.1.2. Receipt of KRB_AS_REQ Message

If all goes well, processing the KRB_AS_REQ message will result in the creation of a ticket for the client to present to the server. The format for the ticket is described in Section 5.3.

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 processed successfully, the KDC MUST respond with a KRB_AS_REP message rather than a replay error. In order to reduce 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.

3.1.3. Generation of KRB_AS_REP Message

The authentication server looks up the client and server principals named in the KRB_AS_REQ in its database, extracting their respective keys. If the requested client principal named in the request is unknown because it doesn't exist in the KDC's principal database, then an error message with a KDC_ERR_C_PRINCIPAL_UNKNOWN is returned.

If required to do so, the server pre-authenticates the request, and if the pre-authentication check fails, an error message with the code KDC_ERR_PREAUTH_FAILED is returned. If pre-authentication is

required, but was not present in the request, an error message with the code KDC_ERR_PREAUTH_REQUIRED is returned, and a METHOD-DATA object will be stored in the e-data field of the KRB-ERROR message to specify which pre-authentication mechanisms are acceptable. Usually this will include PA-ETYPE-INFO and/or PA-ETYPE-INFO2 elements as described below. If the server cannot accommodate any encryption type requested by the client, an error message with code KDC_ERR_ETYPE_NOSUPP is returned. Otherwise, the KDC generates a 'random' session key, meaning that, among other things, it should be impossible to guess the next session key based on knowledge of past session keys. Although this can be achieved in a pseudo-random number generator if it is based on cryptographic principles, it is more desirable to use a truly random number generator, such as one based on measurements of random physical phenomena. See [RFC4086] for an in-depth discussion of randomness.

In response to an AS request, if there are multiple encryption keys registered for a client in the Kerberos database, then the etype field from the AS request is used by the KDC to select the encryption method to be used to protect the encrypted part of the KRB_AS_REP message that is sent to the client. If there is more than one supported strong encryption type in the etype list, the KDC SHOULD use the first valid strong etype for which an encryption key is available.

When the user's key is generated from a password or pass phrase, the string-to-key function for the particular encryption key type is used, as specified in [RFC3961]. The salt value and additional parameters for the string-to-key function have default values (specified by Section 4 and by the encryption mechanism specification, respectively) that may be overridden by pre-authentication data (PA-PW-SALT, PA-AFS3-SALT, PA-ETYPE-INFO, PA-ETYPE-INFO2, etc). Since the KDC is presumed to store a copy of the resulting key only, these values should not be changed for password-based keys except when changing the principal's key.

When the AS server is to include pre-authentication data in a KRB-ERROR or in an AS-REP, it MUST use PA-ETYPE-INFO2, not PA-ETYPE-INFO, if the etype field of the client's AS-REQ lists at least one "newer" encryption type. Otherwise (when the etype field of the client's AS-REQ does not list any "newer" encryption types), it MUST send both PA-ETYPE-INFO2 and PA-ETYPE-INFO (both with an entry for each enctype). A "newer" enctype is any enctype first officially specified concurrently with or subsequent to the issue of this RFC. The enctypes DES, 3DES, or RC4 and any defined in [RFC1510] are not "newer" enctypes.

It is not possible to generate a user's key reliably given a pass phrase without contacting the KDC, since it will not be known whether alternate salt or parameter values are required.

The KDC will attempt to assign the type of the random session key from the list of methods in the etype field. The KDC will select the appropriate type using the list of methods provided and information from the Kerberos database indicating acceptable encryption methods for the application server. The KDC will not issue tickets with a weak session key encryption type.

If the requested starttime is absent, indicates a time in the past, or is within the window of acceptable clock skew for the KDC and the POSTDATE option has not been specified, then the starttime of the ticket is set to the authentication server's current time. If it indicates a time in the future beyond the acceptable clock skew, but the POSTDATED option has not been specified, then the error KDC_ERR_CANNOT_POSTDATE is returned. Otherwise the requested starttime is checked against the policy of the local realm (the administrator might decide to prohibit certain types or ranges of postdated tickets), and if the ticket's starttime is acceptable, it is set as requested, and the INVALID flag is set in the new ticket. The postdated ticket MUST be validated before use by presenting it to the KDC after the starttime has been reached.

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

* The expiration time (endtime) requested in the KRB_AS_REQ message.

* The ticket's starttime plus the maximum allowable lifetime associated with the client principal from the authentication server's database.

* The ticket's starttime plus the maximum allowable lifetime associated with the server principal.

* The ticket's starttime plus the maximum lifetime set by the policy of the local realm.

If the requested expiration time minus the starttime (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 (the field and option names are described fully in Section 5.4.1).

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 starttime of the ticket plus the minimum of the two maximum renewable lifetimes associated with the principals' database entries.

* The starttime of the ticket plus the maximum renewable lifetime set by the policy of the local realm.

The flags field of the new ticket will have the following options set if they have been requested and if the policy of the local realm allows: FORWARDABLE, MAY-POSTDATE, POSTDATED, PROXIABLE, RENEWABLE. If the new ticket is postdated (the starttime is in the future), its INVALID flag will also be set.

If all of the above succeed, the server will encrypt the ciphertext part of the ticket using the encryption key extracted from the server principal's record in the Kerberos database using the encryption type associated with the server principal's key. (This choice is NOT affected by the etype field in the request.) It then formats a KRB_AS_REP message (see Section 5.4.2), copying the addresses in the request into the caddr of the response, placing any required pre-authentication data into the padata of the response, and encrypts the ciphertext part in the client's key using an acceptable encryption method requested in the etype field of the request, or in some key specified by pre-authentication mechanisms being used.

3.1.4. Generation of KRB_ERROR Message

Several errors can occur, and the Authentication Server responds by returning an error message, KRB_ERROR, to the client, with the error-code and e-text fields set to appropriate values. The error message contents and details are described in Section 5.9.1.

3.1.5. Receipt of KRB_AS_REP Message

If the reply message type is KRB_AS_REP, then the client verifies that the cname and crealm fields in the cleartext portion of the reply match what it requested. If any padata fields are present, they may be used to derive the proper secret key to decrypt the

message. The client decrypts the encrypted part of the response using its secret key and verifies that the nonce in the encrypted part matches the nonce it supplied in its request (to detect replays). It also verifies that the sname and srealm in the response match those in the request (or are otherwise expected values), and that the host address field is also correct. It then stores the ticket, session key, start and expiration times, and other information for later use. The last-req field (and the deprecated key-expiration field) from the encrypted part of the response MAY be checked to notify the user of impending key expiration. This enables the client program to suggest remedial action, such as a password change.

Upon validation of the KRB_AS_REP message (by checking the returned nonce against that sent in the KRB_AS_REQ message), the client knows that the current time on the KDC is that read from the authtime field of the encrypted part of the reply. The client can optionally use this value for clock synchronization in subsequent messages by recording with the ticket the difference (offset) between the authtime value and the local clock. This offset can then be used by the same user to adjust the time read from the system clock when generating messages [DGT96].

This technique MUST be used when adjusting for clock skew instead of directly changing the system clock, because the KDC reply is only authenticated to the user whose secret key was used, but not to the system or workstation. If the clock were adjusted, an attacker colluding with a user logging into a workstation could agree on a password, resulting in a KDC reply that would be correctly validated even though it did not originate from a KDC trusted by the workstation.

Proper decryption of the KRB_AS_REP message is not sufficient for the host to verify the identity of the user; the user and an attacker could cooperate to generate a KRB_AS_REP format message that decrypts properly but is not from the proper KDC. If the host wishes to verify the identity of the user, it MUST require the user to present application credentials that can be verified using a securely-stored secret key for the host. If those credentials can be verified, then the identity of the user can be assured.

3.1.6. Receipt of KRB_ERROR Message

If the reply message type is KRB_ERROR, then the client interprets it as an error and performs whatever application-specific tasks are necessary for recovery.

3.2. The Client/Server Authentication Exchange

Summary

Message direction	Message type	Section
Client to Application server	KRB_AP_REQ	5.5.1
[optional] Application server to client	KRB_AP_REP or KRB_ERROR	5.5.2 5.9.1

The client/server authentication (CS) exchange is used by network applications to authenticate the client to the server and vice versa. The client MUST have already acquired credentials for the server using the AS or TGS exchange.

3.2.1. The KRB_AP_REQ Message

The KRB_AP_REQ contains authentication information that SHOULD be part of the first message in an authenticated transaction. It contains a ticket, an authenticator, and some additional bookkeeping information (see Section 5.5.1 for the exact format). The ticket by itself is insufficient to authenticate a client, since tickets are passed across the network in cleartext (tickets contain both an encrypted and unencrypted portion, so cleartext here refers to the entire unit, which can be copied from one message and replayed in another without any cryptographic skill). The authenticator is used to prevent invalid replay of tickets by proving to the server that the client knows the session key of the ticket and thus is entitled to use the ticket. The KRB_AP_REQ message is referred to elsewhere as the 'authentication header'.

3.2.2. Generation of a KRB_AP_REQ Message

When a client wishes to initiate authentication to a server, it obtains (either through a credentials cache, the AS exchange, or the TGS exchange) a ticket and session key for the desired service. The client MAY re-use any tickets it holds until they expire. To use a ticket, the client constructs a new Authenticator from the system time and its name, and optionally from an application-specific checksum, an initial sequence number to be used in KRB_SAFE or KRB_PRIV messages, and/or a session subkey to be used in negotiations for a session key unique to this particular session. Authenticators MUST NOT be re-used and SHOULD be rejected if replayed to a server. Note that this can make applications based on unreliable transports difficult to code correctly. If the transport might deliver duplicated messages, either a new authenticator MUST be generated for each retry, or the application server MUST match requests and replies and replay the first reply in response to a detected duplicate.

If a sequence number is to be included, it SHOULD be randomly chosen so that even after many messages have been exchanged it is not likely to collide with other sequence numbers in use.

The client MAY indicate a requirement of mutual authentication or the use of a session-key based ticket (for user-to-user authentication, see section 3.7) by setting the appropriate flag(s) in the ap-options field of the message.

The Authenticator is encrypted in the session key and combined with the ticket to form the KRB_AP_REQ message, which is then sent to the end server along with any additional application-specific information.

3.2.3. Receipt of KRB_AP_REQ Message

Authentication is based on the server's current time of day (clocks MUST be loosely synchronized), the authenticator, and the ticket. Several errors are possible. If an error occurs, the server is expected to reply to the client with a KRB_ERROR message. This message MAY be encapsulated in the application protocol if its raw form is not acceptable to the protocol. The format of error messages is described in Section 5.9.1.

The algorithm for verifying authentication information is as follows. If the message type is not KRB_AP_REQ, the server returns the KRB_AP_ERR_MSG_TYPE error. If the key version indicated by the Ticket in the KRB_AP_REQ is not one the server can use (e.g., it indicates an old key, and the server no longer possesses a copy of the old key), the KRB_AP_ERR_BADKEYVER error is returned. If the USE-SESSION-KEY flag is set in the ap-options field, it indicates to the server that user-to-user authentication is in use, and that the ticket is encrypted in the session key from the server's TGT rather than in the server's secret key. See Section 3.7 for a more complete description of the effect of user-to-user authentication on all messages in the Kerberos protocol.

Because it is possible for the server to be registered in multiple realms, with different keys in each, the srealm field in the unencrypted portion of the ticket in the KRB_AP_REQ is used to specify which secret key the server should use to decrypt that ticket. The KRB_AP_ERR_NOKEY error code is returned if the server doesn't have the proper key to decipher the ticket.

The ticket is decrypted using the version of the server's key specified by the ticket. If the decryption routines detect a modification of the ticket (each encryption system MUST provide safeguards to detect modified ciphertext), the

KRB_AP_ERR_BAD_INTEGRITY error is returned (chances are good that different keys were used to encrypt and decrypt).

The authenticator is decrypted using the session key extracted from the decrypted ticket. If decryption shows that is has been modified, the KRB_AP_ERR_BAD_INTEGRITY error is returned. The name and realm of the client from the ticket are compared against the same fields in the authenticator. If they don't match, the KRB_AP_ERR_BADMATCH error is returned; normally this is caused by a client error or an attempted attack. The addresses in the ticket (if any) are then searched for an address matching the operating-system reported address of the client. If no match is found or the server insists on ticket addresses but none are present in the ticket, the KRB_AP_ERR_BADADDR error is returned. If the local (server) time and the client time in the authenticator differ by more than the allowable clock skew (e.g., 5 minutes), the KRB_AP_ERR_SKEW error is returned.

Unless the application server provides its own suitable means to protect against replay (for example, a challenge-response sequence initiated by the server after authentication, or use of a server-generated encryption subkey), the server MUST utilize a replay cache to remember any authenticator presented within the allowable clock skew. Careful analysis of the application protocol and implementation is recommended before eliminating this cache. The replay cache will store at least the server name, along with the client name, time, and microsecond fields from the recently-seen authenticators, and if a matching tuple is found, the KRB_AP_ERR_REPEAT error is returned. Note that the rejection here is restricted to authenticators from the same principal to the same server. Other client principals communicating with the same server principal should not have their authenticators rejected if the time and microsecond fields happen to match some other client's authenticator.

If a server loses track of authenticators presented within the allowable clock skew, it MUST reject all requests until the clock skew interval has passed, providing assurance that any lost or replayed authenticators will fall outside the allowable clock skew and can no longer be successfully replayed. If this were not done, an attacker could subvert the authentication by recording the ticket and authenticator sent over the network to a server and replaying them following an event that caused the server to lose track of recently seen authenticators.

Implementation note: If a client generates multiple requests to the KDC with the same timestamp, including the microsecond field, all but the first of the requests received will be rejected as replays. This

might happen, for example, if the resolution of the client's clock is too coarse. Client implementations SHOULD ensure that the timestamps are not reused, possibly by incrementing the microseconds field in the time stamp when the clock returns the same time for multiple requests.

If multiple servers (for example, different services on one machine, or a single service implemented on multiple machines) share a service principal (a practice that we do not recommend in general, but that we acknowledge will be used in some cases), either they MUST share this replay cache, or the application protocol MUST be designed so as to eliminate the need for it. Note that this applies to all of the services. If any of the application protocols does not have replay protection built in, an authenticator used with such a service could later be replayed to a different service with the same service principal but no replay protection, if the former doesn't record the authenticator information in the common replay cache.

If a sequence number is provided in the authenticator, the server saves it for later use in processing KRB_SAFE and/or KRB_PRIV messages. If a subkey is present, the server either saves it for later use or uses it to help generate its own choice for a subkey to be returned in a KRB_AP_REP message.

The server computes the age of the ticket: local (server) time minus the starttime inside the Ticket. If the starttime is later than the current time by more than the allowable clock skew, or if the INVALID flag is set in the ticket, the KRB_AP_ERR_TKT_NYV error is returned. Otherwise, if the current time is later than end time by more than the allowable clock skew, the KRB_AP_ERR_TKT_EXPIRED error is returned.

If all these checks succeed without an error, the server is assured that the client possesses the credentials of the principal named in the ticket, and thus, that the client has been authenticated to the server.

Passing these checks provides only authentication of the named principal; it does not imply authorization to use the named service. Applications MUST make a separate authorization decision based upon the authenticated name of the user, the requested operation, local access control information such as that contained in a .k5login or .k5users file, and possibly a separate distributed authorization service.

3.2.4. Generation of a KRB_AP_REP Message

Typically, a client's request will include both the authentication information and its initial request in the same message, and the server need not explicitly reply to the KRB_AP_REQ. However, if mutual authentication (authenticating not only the client to the server, but also the server to the client) is being performed, the KRB_AP_REQ message will have MUTUAL-REQUIRED set in its ap-options field, and a KRB_AP_REP message is required in response. As with the error message, this message MAY be encapsulated in the application protocol if its "raw" form is not acceptable to the application's protocol. The timestamp and microsecond field used in the reply MUST be the client's timestamp and microsecond field (as provided in the authenticator). If a sequence number is to be included, it SHOULD be randomly chosen as described above for the authenticator. A subkey MAY be included if the server desires to negotiate a different subkey. The KRB_AP_REP message is encrypted in the session key extracted from the ticket.

Note that in the Kerberos Version 4 protocol, the timestamp in the reply was the client's timestamp plus one. This is not necessary in Version 5 because Version 5 messages are formatted in such a way that it is not possible to create the reply by judicious message surgery (even in encrypted form) without knowledge of the appropriate encryption keys.

3.2.5. Receipt of KRB_AP_REP Message

If a KRB_AP_REP message is returned, the client uses the session key from the credentials obtained for the server to decrypt the message and verifies that the timestamp and microsecond fields match those in the Authenticator it sent to the server. If they match, then the client is assured that the server is genuine. The sequence number and subkey (if present) are retained for later use. (Note that for encrypting the KRB_AP_REP message, the sub-session key is not used, even if it is present in the Authentication.)

3.2.6. Using the Encryption Key

After the KRB_AP_REQ/KRB_AP_REP exchange has occurred, the client and server share an encryption key that can be used by the application. In some cases, the use of this session key will be implicit in the protocol; in others the method of use must be chosen from several alternatives. The application MAY choose the actual encryption key to be used for KRB_PRIV, KRB_SAFE, or other application-specific uses based on the session key from the ticket and subkeys in the KRB_AP_REP message and the authenticator. Implementations of the protocol MAY provide routines to choose subkeys based on session keys

and random numbers and to generate a negotiated key to be returned in the KRB_AP_REP message.

To mitigate the effect of failures in random number generation on the client, it is strongly encouraged that any key derived by an application for subsequent use include the full key entropy derived from the KDC-generated session key carried in the ticket. We leave the protocol negotiations of how to use the key (e.g., for selecting an encryption or checksum type) to the application programmer. The Kerberos protocol does not constrain the implementation options, but an example of how this might be done follows.

One way that an application may choose to negotiate a key to be used for subsequent integrity and privacy protection is for the client to propose a key in the subkey field of the authenticator. The server can then choose a key using the key proposed by the client as input, returning the new subkey in the subkey field of the application reply. This key could then be used for subsequent communication.

With both the one-way and mutual authentication exchanges, the peers should take care not to send sensitive information to each other without proper assurances. In particular, applications that require privacy or integrity SHOULD use the KRB_AP_REP response from the server to the client to assure both client and server of their peer's identity. If an application protocol requires privacy of its messages, it can use the KRB_PRIV message (section 3.5). The KRB_SAFE message (Section 3.4) can be used to ensure integrity.

3.3. The Ticket-Granting Service (TGS) Exchange

Summary

Message direction	Message type	Section
1. Client to Kerberos	KRB_TGS_REQ	5.4.1
2. Kerberos to client	KRB_TGS_REP or KRB_ERROR	5.4.2 5.9.1

The TGS exchange between a client and the Kerberos TGS is initiated by a client when it seeks to obtain authentication credentials for a given server (which might be registered in a remote realm), when it seeks to renew or validate an existing ticket, or when it seeks to obtain a proxy ticket. In the first case, the client must already have acquired a ticket for the Ticket-Granting Service using the AS exchange (the TGT is usually obtained when a client initially authenticates to the system, such as when a user logs in). The message format for the TGS exchange is almost identical to that for the AS exchange. The primary difference is that encryption and decryption in the TGS exchange does not take place under the client's

key. Instead, the session key from the TGT or renewable ticket, or sub-session key from an Authenticator is used. As is the case for all application servers, expired tickets are not accepted by the TGS, so once a renewable or TGT expires, the client must use a separate exchange to obtain valid tickets.

The TGS exchange consists of two messages: a request (KRB_TGS_REQ) from the client to the Kerberos Ticket-Granting Server, and a reply (KRB_TGS_REP or KRB_ERROR). The KRB_TGS_REQ message includes information authenticating the client plus a request for credentials. The authentication information consists of the authentication header (KRB_AP_REQ), which includes the client's previously obtained ticket-granting, renewable, or invalid ticket. In the TGT and proxy cases, the request MAY include one or more of the following: a list of network addresses, a collection of typed authorization data to be sealed in the ticket for authorization use by the application server, or additional tickets (the use of which are described later). The TGS reply (KRB_TGS_REP) contains the requested credentials, encrypted in the session key from the TGT or renewable ticket, or, if present, in the sub-session key from the Authenticator (part of the authentication header). The KRB_ERROR message contains an error code and text explaining what went wrong. The KRB_ERROR message is not encrypted. The KRB_TGS_REP message contains information that can be used to detect replays, and to associate it with the message to which it replies. The KRB_ERROR message also contains information that can be used to associate it with the message to which it replies. The same comments about integrity protection of KRB_ERROR messages mentioned in Section 3.1 apply to the TGS exchange.

3.3.1. Generation of KRB_TGS_REQ Message

Before sending a request to the ticket-granting service, the client MUST determine in which realm the application server is believed to be registered. This can be accomplished in several ways. It might be known beforehand (since the realm is part of the principal identifier), it might be stored in a nameserver, or it might be obtained from a configuration file. If the realm to be used is obtained from a nameserver, there is a danger of being spoofed if the nameservice providing the realm name is not authenticated. This might result in the use of a realm that has been compromised, which would result in an attacker's ability to compromise the authentication of the application server to the client.

If the client knows the service principal name and realm and it does not already possess a TGT for the appropriate realm, then one must be obtained. This is first attempted by requesting a TGT for the destination realm from a Kerberos server for which the client possesses a TGT (by using the KRB_TGS_REQ message recursively). The

Kerberos server MAY return a TGT for the desired realm, in which case one can proceed. Alternatively, the Kerberos server MAY return a TGT for a realm that is 'closer' to the desired realm (further along the standard hierarchical path between the client's realm and the requested realm server's realm). Note that in this case misconfiguration of the Kerberos servers may cause loops in the resulting authentication path, which the client should be careful to detect and avoid.

If the Kerberos server returns a TGT for a realm 'closer' than the desired realm, the client MAY use local policy configuration to verify that the authentication path used is an acceptable one. Alternatively, a client MAY choose its own authentication path, rather than rely on the Kerberos server to select one. In either case, any policy or configuration information used to choose or validate authentication paths, whether by the Kerberos server or by the client, MUST be obtained from a trusted source.

When a client obtains a TGT that is 'closer' to the destination realm, the client MAY cache this ticket and reuse it in future KRB-TGS exchanges with services in the 'closer' realm. However, if the client were to obtain a TGT for the 'closer' realm by starting at the initial KDC rather than as part of obtaining another ticket, then a shorter path to the 'closer' realm might be used. This shorter path may be desirable because fewer intermediate KDCs would know the session key of the ticket involved. For this reason, clients SHOULD evaluate whether they trust the realms transited in obtaining the 'closer' ticket when making a decision to use the ticket in future.

Once the client obtains a TGT for the appropriate realm, it determines which Kerberos servers serve that realm and contacts one of them. The list might be obtained through a configuration file or network service, or it MAY be generated from the name of the realm. As long as the secret keys exchanged by realms are kept secret, only denial of service results from using a false Kerberos server.

As in the AS exchange, the client MAY specify a number of options in the KRB_TGS_REQ message. One of these options is the ENC-TKT-IN-SKEY option used for user-to-user authentication. An overview of user-to-user authentication can be found in Section 3.7. When generating the KRB_TGS_REQ message, this option indicates that the client is including a TGT obtained from the application server in the additional tickets field of the request and that the KDC SHOULD encrypt the ticket for the application server using the session key from this additional ticket, instead of a server key from the principal database.

The client prepares the KRB_TGS_REQ message, providing an authentication header as an element of the padata field, and including the same fields as used in the KRB_AS_REQ message along with several optional fields: the enc-authorizatfion-data field for application server use and additional tickets required by some options.

In preparing the authentication header, the client can select a sub-session key under which the response from the Kerberos server will be encrypted. If the client selects a sub-session key, care must be taken to ensure the randomness of the selected sub-session key.

If the sub-session key is not specified, the session key from the TGT will be used. If the enc-authorization-data is present, it MUST be encrypted in the sub-session key, if present, from the authenticator portion of the authentication header, or, if not present, by using the session key from the TGT.

Once prepared, the message is sent to a Kerberos server for the destination realm.

3.3.2. Receipt of KRB_TGS_REQ Message

The KRB_TGS_REQ message is processed in a manner similar to the KRB_AS_REQ message, but there are many additional checks to be performed. First, the Kerberos server MUST determine which server the accompanying ticket is for, and it MUST select the appropriate key to decrypt it. For a normal KRB_TGS_REQ message, it will be for the ticket-granting service, and the TGS's key will be used. If the TGT was issued by another realm, then the appropriate inter-realm key MUST be used. If (a) the accompanying ticket is not a TGT for the current realm, but is for an application server in the current realm, (b) the RENEW, VALIDATE, or PROXY options are specified in the request, and (c) the server for which a ticket is requested is the server named in the accompanying ticket, then the KDC will decrypt the ticket in the authentication header using the key of the server for which it was issued. If no ticket can be found in the padata field, the KDC_ERR_PADATA_TYPE_NOSUPP error is returned.

Once the accompanying ticket has been decrypted, the user-supplied checksum in the Authenticator MUST be verified against the contents of the request, and the message MUST be rejected if the checksums do not match (with an error code of KRB_AP_ERR_MODIFIED) or if the checksum is not collision-proof (with an error code of KRB_AP_ERR_INAPP_CKSUM). If the checksum type is not supported, the KDC_ERR_SUMTYPE_NOSUPP error is returned. If the authorization-data are present, they are decrypted using the sub-session key from the Authenticator.

If any of the decryptions indicate failed integrity checks, the KRB_AP_ERR_BAD_INTEGRITY error is returned.

As discussed in Section 3.1.2, the KDC MUST send a valid KRB_TGS_REP message if it receives a KRB_TGS_REQ message identical to one it has recently processed. However, if the authenticator is a replay, but the rest of the request is not identical, then the KDC SHOULD return KRB_AP_ERR_REPEAT.

3.3.3. Generation of KRB_TGS_REP Message

The KRB_TGS_REP message shares its format with the KRB_AS_REP (KRB_KDC_REP), but with its type field set to KRB_TGS_REP. The detailed specification is in Section 5.4.2.

The response will include a ticket for the requested server or for a ticket granting server of an intermediate KDC to be contacted to obtain the requested ticket. The Kerberos database is queried to retrieve the record for the appropriate server (including the key with which the ticket will be encrypted). If the request is for a TGT for a remote realm, and if no key is shared with the requested realm, then the Kerberos server will select the realm 'closest' to the requested realm with which it does share a key and use that realm instead. This is the only case where the response for the KDC will be for a different server than that requested by the client.

By default, the address field, the client's name and realm, the list of transited realms, the time of initial authentication, the expiration time, and the authorization data of the newly-issued ticket will be copied from the TGT or renewable ticket. If the transited field needs to be updated, but the transited type is not supported, the KDC_ERR_TRTYPE_NOSUPP error is returned.

If the request specifies an endtime, then the endtime of the new ticket is set to the minimum of (a) that request, (b) the endtime from the TGT, and (c) the starttime of the TGT plus the minimum of the maximum life for the application server and the maximum life for the local realm (the maximum life for the requesting principal was already applied when the TGT was issued). If the new ticket is to be a renewal, then the endtime above is replaced by the minimum of (a) the value of the renew_till field of the ticket and (b) the starttime for the new ticket plus the life (endtime-starttime) of the old ticket.

If the FORWARDED option has been requested, then the resulting ticket will contain the addresses specified by the client. This option will only be honored if the FORWARDABLE flag is set in the TGT. The PROXY option is similar; the resulting ticket will contain the addresses

specified by the client. It will be honored only if the PROXIABLE flag in the TGT is set. The PROXY option will not be honored on requests for additional TGTs.

If the requested starttime is absent, indicates a time in the past, or is within the window of acceptable clock skew for the KDC and the POSTDATE option has not been specified, then the starttime of the ticket is set to the authentication server's current time. If it indicates a time in the future beyond the acceptable clock skew, but the POSTDATED option has not been specified or the MAY-POSTDATE flag is not set in the TGT, then the error KDC_ERR_CANNOT_POSTDATE is returned. Otherwise, if the TGT has the MAY-POSTDATE flag set, then the resulting ticket will be postdated, and the requested starttime is checked against the policy of the local realm. If acceptable, the ticket's starttime is set as requested, and the INVALID flag is set. The postdated ticket MUST be validated before use by presenting it to the KDC after the starttime has been reached. However, in no case may the starttime, endtime, or renew-till time of a newly-issued postdated ticket extend beyond the renew-till time of the TGT.

If the ENC-TKT-IN-SKEY option has been specified and an additional ticket has been included in the request, it indicates that the client is using user-to-user authentication to prove its identity to a server that does not have access to a persistent key. Section 3.7 describes the effect of this option on the entire Kerberos protocol. When generating the KRB_TGS_REP message, this option in the KRB_TGS_REQ message tells the KDC to decrypt the additional ticket using the key for the server to which the additional ticket was issued and to verify that it is a TGT. If the name of the requested server is missing from the request, the name of the client in the additional ticket will be used. Otherwise, the name of the requested server will be compared to the name of the client in the additional ticket. If it is different, the request will be rejected. If the request succeeds, the session key from the additional ticket will be used to encrypt the new ticket that is issued instead of using the key of the server for which the new ticket will be used.

If (a) the name of the server in the ticket that is presented to the KDC as part of the authentication header is not that of the TGS itself, (b) the server is registered in the realm of the KDC, and (c) the RENEW option is requested, then the KDC will verify that the RENEWABLE flag is set in the ticket, that the INVALID flag is not set in the ticket, and that the renew_till time is still in the future. If the VALIDATE option is requested, the KDC will check that the starttime has passed and that the INVALID flag is set. If the PROXY option is requested, then the KDC will check that the PROXIABLE flag

is set in the ticket. If the tests succeed and the ticket passes the hotlist check described in the next section, the KDC will issue the appropriate new ticket.

The ciphertext part of the response in the KRB_TGS_REP message is encrypted in the sub-session key from the Authenticator, if present, or in the session key from the TGT. It is not encrypted using the client's secret key. Furthermore, the client's key's expiration date and the key version number fields are left out since these values are stored along with the client's database record, and that record is not needed to satisfy a request based on a TGT.

3.3.3.1. Checking for Revoked Tickets

Whenever a request is made to the ticket-granting server, the presented ticket(s) is (are) checked against a hot-list of tickets that 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 TGT 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 tickets require no interaction with the KDC), but only until its 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.

3.3.3.2. Encoding the Transited Field

If the identity of the server in the TGT that is presented to the KDC as part of the authentication header is that of the ticket-granting service, but the TGT was issued from another realm, the KDC will look up the inter-realm key shared with that realm and use that key to decrypt the ticket. If the ticket is valid, then the KDC will honor the request, subject to the constraints outlined above in the section describing the AS exchange. The realm part of the client's identity will be taken from the TGT. The name of the realm that issued the TGT, if it is not the realm of the client principal, will be added to the transited field of the ticket to be issued. This is accomplished by reading the transited field from the TGT (which is treated as an unordered set of realm names), adding the new realm to the set, and then constructing and writing out its encoded (shorthand) form (this may involve a rearrangement of the existing encoding).

Note that the ticket-granting service does not add the name of its own realm. Instead, its responsibility is to add the name of the

previous realm. This prevents a malicious Kerberos server from intentionally leaving out its own name (it could, however, omit other realms' names).

The names of neither the local realm nor the principal's realm are to be included in the transited field. They appear elsewhere in the ticket and both are known to have taken part in authenticating the principal. Because the endpoints are not included, both local and single-hop inter-realm authentication result in a transited field that is empty.

Because this field has the name of each transited realm added to it, it might potentially be very long. To decrease the length of this field, its contents are encoded. The initially supported encoding is optimized for the normal case of inter-realm communication: a hierarchical arrangement of realms using either domain or X.500 style realm names. This encoding (called DOMAIN-X500-COMPRESS) is now described.

Realm names in the transited field are separated by a ",". The ",", "\", trailing "."s, and leading spaces (" ") are special characters, and if they are part of a realm name, they MUST be quoted in the transited field by preceding them with a "\".

A realm name ending with a "." is interpreted as being prepended to the previous realm. For example, we can encode traversal of EDU, MIT.EDU, ATHENA.MIT.EDU, WASHINGTON.EDU, and CS.WASHINGTON.EDU as:

```
"EDU,MIT.,ATHENA.,WASHINGTON.EDU,CS.".
```

Note that if either ATHENA.MIT.EDU, or CS.WASHINGTON.EDU were endpoints, they would not be included in this field, and we would have:

```
"EDU,MIT.,WASHINGTON.EDU"
```

A realm name beginning with a "/" is interpreted as being appended to the previous realm. For the purpose of appending, the realm preceding the first listed realm is considered the null realm (""). If a realm name beginning with a "/" is to stand by itself, then it SHOULD be preceded by a space (" "). For example, we can encode traversal of /COM/HP/APOLLO, /COM/HP, /COM, and /COM/DEC as:

```
"/COM,/HP,/APOLLO, /COM/DEC".
```

As in the example above, if /COM/HP/APOLLO and /COM/DEC were endpoints, they would not be included in this field, and we would have:

"/COM,/HP"

A null subfield preceding or following a "," indicates that all realms between the previous realm and the next realm have been traversed. For the purpose of interpreting null subfields, the client's realm is considered to precede those in the transited field, and the server's realm is considered to follow them. Thus, "," means that all realms along the path between the client and the server have been traversed. ",EDU, /COM," means that all realms from the client's realm up to EDU (in a domain style hierarchy) have been traversed, and that everything from /COM down to the server's realm in an X.500 style has also been traversed. This could occur if the EDU realm in one hierarchy shares an inter-realm key directly with the /COM realm in another hierarchy.

3.3.4. Receipt of KRB_TGS_REP Message

When the KRB_TGS_REP is received by the client, it is processed in the same manner as the KRB_AS_REP processing described above. The primary difference is that the ciphertext part of the response must be decrypted using the sub-session key from the Authenticator, if it was specified in the request, or the session key from the TGT, rather than the client's secret key. The server name returned in the reply is the true principal name of the service.

3.4. The KRB_SAFE Exchange

The KRB_SAFE message MAY be used by clients requiring the ability to detect modifications of messages they exchange. It achieves this by including a keyed collision-proof checksum of the user data and some control information. The checksum is keyed with an encryption key (usually the last key negotiated via subkeys, or the session key if no negotiation has occurred).

3.4.1. Generation of a KRB_SAFE Message

When an application wishes to send a KRB_SAFE message, it collects its data and the appropriate control information and computes a checksum over them. The checksum algorithm should be the keyed checksum mandated to be implemented along with the crypto system used for the sub-session or session key. The checksum is generated using the sub-session key, if present, or the session key. Some implementations use a different checksum algorithm for the KRB_SAFE messages, but doing so in an interoperable manner is not always possible.

The control information for the KRB_SAFE message includes both a timestamp and a sequence number. The designer of an application

using the KRB_SAFE message MUST choose at least one of the two mechanisms. This choice SHOULD be based on the needs of the application protocol.

Sequence numbers are useful when all messages sent will be received by one's peer. Connection state is presently required to maintain the session key, so maintaining the next sequence number should not present an additional problem.

If the application protocol is expected to tolerate lost messages without their being resent, the use of the timestamp is the appropriate replay detection mechanism. Using timestamps is also the appropriate mechanism for multi-cast protocols in which all of one's peers share a common sub-session key, but some messages will be sent to a subset of one's peers.

After computing the checksum, the client then transmits the information and checksum to the recipient in the message format specified in Section 5.6.1.

3.4.2. Receipt of KRB_SAFE Message

When an application receives a KRB_SAFE message, it verifies it as follows. If any error occurs, an error code is reported for use by the application.

The message is first checked by verifying that the protocol version and type fields match the current version and KRB_SAFE, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error. The application verifies that the checksum used is a collision-proof keyed checksum that uses keys compatible with the sub-session or session key as appropriate (or with the application key derived from the session or sub-session keys). If it is not, a KRB_AP_ERR_INAPP_CKSUM error is generated. The sender's address MUST be included in the control information; the recipient verifies that the operating system's report of the sender's address matches the sender's address in the message, and (if a recipient address is specified or the recipient requires an address) that one of the recipient's addresses appears as the recipient's address in the message. To work with network address translation, senders MAY use the directional address type specified in Section 8.1 for the sender address and not include recipient addresses. A failed match for either case generates a KRB_AP_ERR_BADADDR error. Then the timestamp and usec and/or the sequence number fields are checked. If timestamp and usec are expected and not present, or if they are present but not current, the KRB_AP_ERR_SKEW error is generated. Timestamps are not required to be strictly ordered; they are only required to be in the skew window. If the server name, along with the client name, time,

and microsecond fields from the Authenticator match any recently-seen (sent or received) such tuples, the KRB_AP_ERR_REPEAT error is generated. If an incorrect sequence number is included, or if a sequence number is expected but not present, the KRB_AP_ERR_BADORDER error is generated. If neither a time-stamp and usec nor a sequence number is present, a KRB_AP_ERR_MODIFIED error is generated. Finally, the checksum is computed over the data and control information, and if it doesn't match the received checksum, a KRB_AP_ERR_MODIFIED error is generated.

If all the checks succeed, the application is assured that the message was generated by its peer and was not modified in transit.

Implementations SHOULD accept any checksum algorithm they implement that has both adequate security and keys compatible with the sub-session or session key. Unkeyed or non-collision-proof checksums are not suitable for this use.

3.5. The KRB_PRIV Exchange

The KRB_PRIV message MAY be used by clients requiring confidentiality and the ability to detect modifications of exchanged messages. It achieves this by encrypting the messages and adding control information.

3.5.1. Generation of a KRB_PRIV Message

When an application wishes to send a KRB_PRIV message, it collects its data and the appropriate control information (specified in Section 5.7.1) and encrypts them under an encryption key (usually the last key negotiated via subkeys, or the session key if no negotiation has occurred). As part of the control information, the client MUST choose to use either a timestamp or a sequence number (or both); see the discussion in Section 3.4.1 for guidelines on which to use. After the user data and control information are encrypted, the client transmits the ciphertext and some 'envelope' information to the recipient.

3.5.2. Receipt of KRB_PRIV Message

When an application receives a KRB_PRIV message, it verifies it as follows. If any error occurs, an error code is reported for use by the application.

The message is first checked by verifying that the protocol version and type fields match the current version and KRB_PRIV, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error. The application then decrypts the ciphertext and processes

the resultant plaintext. If decryption shows that the data has been modified, a KRB_AP_ERR_BAD_INTEGRITY error is generated.

The sender's address MUST be included in the control information; the recipient verifies that the operating system's report of the sender's address matches the sender's address in the message. If a recipient address is specified or the recipient requires an address, then one of the recipient's addresses MUST also appear as the recipient's address in the message. Where a sender's or receiver's address might not otherwise match the address in a message because of network address translation, an application MAY be written to use addresses of the directional address type in place of the actual network address.

A failed match for either case generates a KRB_AP_ERR_BADADDR error. To work with network address translation, implementations MAY use the directional address type defined in Section 7.1 for the sender address and include no recipient address.

Next the timestamp and usec and/or the sequence number fields are checked. If timestamp and usec are expected and not present, or if they are present but not current, the KRB_AP_ERR_SKEW error is generated. If the server name, along with the client name, time, and microsecond fields from the Authenticator match any such recently-seen tuples, the KRB_AP_ERR_REPEAT error is generated. If an incorrect sequence number is included, or if a sequence number is expected but not present, the KRB_AP_ERR_BADORDER error is generated. If neither a time-stamp and usec nor a sequence number is present, a KRB_AP_ERR_MODIFIED error is generated.

If all the checks succeed, the application can assume the message was generated by its peer and was securely transmitted (without intruders seeing the unencrypted contents).

3.6. The KRB_CRED Exchange

The KRB_CRED message MAY be used by clients requiring the ability to send Kerberos credentials from one host to another. It achieves this by sending the tickets together with encrypted data containing the session keys and other information associated with the tickets.

3.6.1. Generation of a KRB_CRED Message

When an application wishes to send a KRB_CRED message, it first (using the KRB_TGS exchange) obtains credentials to be sent to the remote host. It then constructs a KRB_CRED message using the ticket or tickets so obtained, placing the session key needed to use each

ticket in the key field of the corresponding KrbCredInfo sequence of the encrypted part of the KRB_CRED message.

Other information associated with each ticket and obtained during the KRB_TGS exchange is also placed in the corresponding KrbCredInfo sequence in the encrypted part of the KRB_CRED message. The current time and, if they are specifically required by the application, the nonce, s-address, and r-address fields are placed in the encrypted part of the KRB_CRED message, which is then encrypted under an encryption key previously exchanged in the KRB_AP exchange (usually the last key negotiated via subkeys, or the session key if no negotiation has occurred).

Implementation note: When constructing a KRB_CRED message for inclusion in a GSSAPI initial context token, the MIT implementation of Kerberos will not encrypt the KRB_CRED message if the session key is a DES or triple DES key. For interoperability with MIT, the Microsoft implementation will not encrypt the KRB_CRED in a GSSAPI token if it is using a DES session key. Starting at version 1.2.5, MIT Kerberos can receive and decode either encrypted or unencrypted KRB_CRED tokens in the GSSAPI exchange. The Heimdal implementation of Kerberos can also accept either encrypted or unencrypted KRB_CRED messages. Since the KRB_CRED message in a GSSAPI token is encrypted in the authenticator, the MIT behavior does not present a security problem, although it is a violation of the Kerberos specification.

3.6.2. Receipt of KRB_CRED Message

When an application receives a KRB_CRED message, it verifies it. If any error occurs, an error code is reported for use by the application. The message is verified by checking that the protocol version and type fields match the current version and KRB_CRED, respectively. A mismatch generates a KRB_AP_ERR_BADVERSION or KRB_AP_ERR_MSG_TYPE error. The application then decrypts the ciphertext and processes the resultant plaintext. If decryption shows the data to have been modified, a KRB_AP_ERR_BAD_INTEGRITY error is generated.

If present or required, the recipient MAY verify that the operating system's report of the sender's address matches the sender's address in the message, and that one of the recipient's addresses appears as the recipient's address in the message. The address check does not provide any added security, since the address, if present, has already been checked in the KRB_AP_REQ message and there is not any benefit to be gained by an attacker in reflecting a KRB_CRED message back to its originator. Thus, the recipient MAY ignore the address even if it is present in order to work better in Network Address Translation (NAT) environments. A failed match for either case

generates a KRB_AP_ERR_BADADDR error. Recipients MAY skip the address check, as the KRB_CRED message cannot generally be reflected back to the originator. The timestamp and usec fields (and the nonce field, if required) are checked next. If the timestamp and usec are not present, or if they are present but not current, the KRB_AP_ERR_SKEW error is generated.

If all the checks succeed, the application stores each of the new tickets in its credentials cache together with the session key and other information in the corresponding KrbCredInfo sequence from the encrypted part of the KRB_CRED message.

3.7. User-to-User Authentication Exchanges

User-to-User authentication provides a method to perform authentication when the verifier does not have a access to long-term service key. This might be the case when running a server (for example, a window server) as a user on a workstation. In such cases, the server may have access to the TGT obtained when the user logged in to the workstation, but because the server is running as an unprivileged user, it might not have access to system keys. Similar situations may arise when running peer-to-peer applications.

Summary

Message direction	Message type	Sections
0. Message from application server	Not specified	
1. Client to Kerberos	KRB_TGS_REQ	3.3 & 5.4.1
2. Kerberos to client	KRB_TGS_REP or KRB_ERROR	3.3 & 5.4.2 5.9.1
3. Client to application server	KRB_AP_REQ	3.2 & 5.5.1

To address this problem, the Kerberos protocol allows the client to request that the ticket issued by the KDC be encrypted using a session key from a TGT issued to the party that will verify the authentication. This TGT must be obtained from the verifier by means of an exchange external to the Kerberos protocol, usually as part of the application protocol. This message is shown in the summary above as message 0. Note that because the TGT is encrypted in the KDC's secret key, it cannot be used for authentication without possession of the corresponding secret key. Furthermore, because the verifier does not reveal the corresponding secret key, providing a copy of the verifier's TGT does not allow impersonation of the verifier.

Message 0 in the table above represents an application-specific negotiation between the client and server, at the end of which both have determined that they will use user-to-user authentication, and the client has obtained the server's TGT.

Next, the client includes the server's TGT as an additional ticket in its KRB_TGS_REQ request to the KDC (message 1 in the table above) and specifies the ENC-TKT-IN-SKEY option in its request.

If validated according to the instructions in Section 3.3.3, the application ticket returned to the client (message 2 in the table above) will be encrypted using the session key from the additional ticket and the client will note this when it uses or stores the application ticket.

When contacting the server using a ticket obtained for user-to-user authentication (message 3 in the table above), the client MUST specify the USE-SESSION-KEY flag in the ap-options field. This tells the application server to use the session key associated with its TGT to decrypt the server ticket provided in the application request.

4. Encryption and Checksum Specifications

The Kerberos protocols described in this document are designed to encrypt messages of arbitrary sizes, using stream or block encryption ciphers. Encryption is used to prove the identities of the network entities participating in message exchanges. The Key Distribution Center for each realm is trusted by all principals registered in that realm to store a secret key in confidence. Proof of knowledge of this secret key is used to verify the authenticity of a principal.

The KDC uses the principal's secret key (in the AS exchange) or a shared session key (in the TGS exchange) to encrypt responses to ticket requests; the ability to obtain the secret key or session key implies the knowledge of the appropriate keys and the identity of the KDC. The ability of a principal to decrypt the KDC response and to present a Ticket and a properly formed Authenticator (generated with the session key from the KDC response) to a service verifies the identity of the principal; likewise the ability of the service to extract the session key from the Ticket and to prove its knowledge thereof in a response verifies the identity of the service.

[RFC3961] defines a framework for defining encryption and checksum mechanisms for use with Kerberos. It also defines several such mechanisms, and more may be added in future updates to that document.

The string-to-key operation provided by [RFC3961] is used to produce a long-term key for a principal (generally for a user). The default salt string, if none is provided via pre-authentication data, is the concatenation of the principal's realm and name components, in order, with no separators. Unless it is indicated otherwise, the default string-to-key opaque parameter set as defined in [RFC3961] is used.

Encrypted data, keys, and checksums are transmitted using the EncryptedData, EncryptionKey, and Checksum data objects defined in Section 5.2.9. The encryption, decryption, and checksum operations described in this document use the corresponding encryption, decryption, and get_mic operations described in [RFC3961], with implicit "specific key" generation using the "key usage" values specified in the description of each EncryptedData or Checksum object to vary the key for each operation. Note that in some cases, the value to be used is dependent on the method of choosing the key or the context of the message.

Key usages are unsigned 32-bit integers; zero is not permitted. The key usage values for encrypting or checksumming Kerberos messages are indicated in Section 5 along with the message definitions. The key usage values 512-1023 are reserved for uses internal to a Kerberos implementation. (For example, seeding a pseudo-random number generator with a value produced by encrypting something with a session key and a key usage value not used for any other purpose.) Key usage values between 1024 and 2047 (inclusive) are reserved for application use; applications SHOULD use even values for encryption and odd values for checksums within this range. Key usage values are also summarized in a table in Section 7.5.1.

There might exist other documents that define protocols in terms of the RFC 1510 encryption types or checksum types. These documents would not know about key usages. In order that these specifications continue to be meaningful until they are updated, if no key usage values are specified, then key usages 1024 and 1025 must be used to derive keys for encryption and checksums, respectively. (This does not apply to protocols that do their own encryption independent of this framework, by directly using the key resulting from the Kerberos authentication exchange.) New protocols defined in terms of the Kerberos encryption and checksum types SHOULD use their own key usage values.

Unless it is indicated otherwise, no cipher state chaining is done from one encryption operation to another.

Implementation note: Although it is not recommended, some application protocols will continue to use the key data directly, even if only in currently existing protocol specifications. An implementation intended to support general Kerberos applications may therefore need to make key data available, as well as the attributes and operations described in [RFC3961]. One of the more common reasons for directly performing encryption is direct control over negotiation and selection of a "sufficiently strong" encryption algorithm (in the context of a given application). Although Kerberos does not directly provide a facility for negotiating encryption types between the

application client and server, there are approaches for using Kerberos to facilitate this negotiation. For example, a client may request only "sufficiently strong" session key types from the KDC and expect that any type returned by the KDC will be understood and supported by the application server.

5. Message Specifications

The ASN.1 collected here should be identical to the contents of Appendix A. In the case of a conflict, the contents of Appendix A shall take precedence.

The Kerberos protocol is defined here in terms of Abstract Syntax Notation One (ASN.1) [X680], which provides a syntax for specifying both the abstract layout of protocol messages as well as their encodings. Implementors not utilizing an existing ASN.1 compiler or support library are cautioned to understand the actual ASN.1 specification thoroughly in order 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.

Note that in several places, changes to abstract types from RFC 1510 have been made. This is in part to address widespread assumptions that various implementors have made, in some cases resulting in unintentional violations of the ASN.1 standard. These are clearly flagged where they occur. The differences between the abstract types in RFC 1510 and abstract types in this document can cause incompatible encodings to be emitted when certain encoding rules, e.g., the Packed Encoding Rules (PER), are used. This theoretical incompatibility should not be relevant for Kerberos, since Kerberos explicitly specifies the use of the Distinguished Encoding Rules (DER). It might be an issue for protocols seeking to use Kerberos types with other encoding rules. (This practice is not recommended.) With very few exceptions (most notably the usages of BIT STRING), the encodings resulting from using the DER remain identical between the types defined in RFC 1510 and the types defined in this document.

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

```
KerberosV5Spec2 {
        iso(1) identified-organization(3) dod(6) internet(1)
        security(5) kerberosV5(2) modules(4) krb5spec2(2)
} DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- rest of definitions here

END
```

This specifies that the tagging context for the module will be explicit and non-automatic.

Note that in some other publications (such as [RFC1510] and [RFC1964]), the "dod" portion of the object identifier is erroneously specified as having the value "5". In the case of RFC 1964, use of the "correct" OID value would result in a change in the wire protocol; therefore, it remains unchanged for now.

Note that elsewhere in this document, nomenclature for various message types is inconsistent, but it largely follows C language conventions, including use of underscore (_) characters and all-caps spelling of names intended to be numeric constants. Also, in some places, identifiers (especially those referring to constants) are written in all-caps in order to distinguish them from surrounding explanatory text.

The ASN.1 notation does not permit underscores in identifiers, so in actual ASN.1 definitions, underscores are replaced with hyphens (-). Additionally, structure member names and defined values in ASN.1 MUST begin with a lowercase letter, whereas type names MUST begin with an uppercase letter.

5.1. Specific Compatibility Notes on ASN.1

For compatibility purposes, implementors should heed the following specific notes regarding the use of ASN.1 in Kerberos. These notes do not describe deviations from standard usage of ASN.1. The purpose of these notes is instead to describe some historical quirks and non-compliance of various implementations, as well as historical ambiguities, which, although they are valid ASN.1, can lead to confusion during implementation.

5.1.1. ASN.1 Distinguished Encoding Rules

The encoding of Kerberos protocol messages shall obey the Distinguished Encoding Rules (DER) of ASN.1 as described in [X690]. Some implementations (believed primarily to be those derived from DCE 1.1 and earlier) are known to use the more general Basic Encoding

Rules (BER); in particular, these implementations send indefinite encodings of lengths. Implementations MAY accept such encodings in the interest of backward compatibility, though implementors are warned that decoding fully-general BER is fraught with peril.

5.1.2. Optional Integer Fields

Some implementations do not internally distinguish between an omitted optional integer value and a transmitted value of zero. The places in the protocol where this is relevant include various microseconds fields, nonces, and sequence numbers. Implementations SHOULD treat omitted optional integer values as having been transmitted with a value of zero, if the application is expecting this.

5.1.3. Empty SEQUENCE OF Types

There are places in the protocol where a message contains a SEQUENCE OF type as an optional member. This can result in an encoding that contains an empty SEQUENCE OF encoding. The Kerberos protocol does not semantically distinguish between an absent optional SEQUENCE OF type and a present optional but empty SEQUENCE OF type. Implementations SHOULD NOT send empty SEQUENCE OF encodings that are marked OPTIONAL, but SHOULD accept them as being equivalent to an omitted OPTIONAL type. In the ASN.1 syntax describing Kerberos messages, instances of these problematic optional SEQUENCE OF types are indicated with a comment.

5.1.4. Unrecognized Tag Numbers

Future revisions to this protocol may include new message types with different APPLICATION class tag numbers. Such revisions should protect older implementations by only sending the message types to parties that are known to understand them; e.g., by means of a flag bit set by the receiver in a preceding request. In the interest of robust error handling, implementations SHOULD gracefully handle receiving a message with an unrecognized tag anyway, and return an error message, if appropriate.

In particular, KDCs SHOULD return KRB_AP_ERR_MSG_TYPE if the incorrect tag is sent over a TCP transport. The KDCs SHOULD NOT respond to messages received with an unknown tag over UDP transport in order to avoid denial of service attacks. For non-KDC applications, the Kerberos implementation typically indicates an error to the application which takes appropriate steps based on the application protocol.

5.1.5. Tag Numbers Greater Than 30

A naive implementation of a DER ASN.1 decoder may experience problems with ASN.1 tag numbers greater than 30, due to such tag numbers being encoded using more than one byte. Future revisions of this protocol may utilize tag numbers greater than 30, and implementations SHOULD be prepared to gracefully return an error, if appropriate, when they do not recognize the tag.

5.2. Basic Kerberos Types

This section defines a number of basic types that are potentially used in multiple Kerberos protocol messages.

5.2.1. KerberosString

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 [ISO-2022/ECMA-35] to switch character sets, and the default character set that is designated as G0 is the ISO-646/ECMA-6 [ISO-646/ECMA-6] International Reference Version (IRV) (a.k.a. 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 prohibits the designation of character sets as any but the G0 and C0 sets. Unfortunately, this seems to have the side effect of prohibiting the use of ISO-8859 (ISO Latin) [ISO-8859] character sets or any other character sets that utilize a 96-character set, as ISO-2022/ECMA-35 prohibits designating them as the G0 code element. This side effect is being investigated in the ASN.1 standards community.

In practice, many implementations treat GeneralStrings as if they were 8-bit strings of whichever character set the implementation defaults to, without regard to 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 adhere 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 new (post-RFC 1510) type KerberosString, defined below, is a GeneralString that is constrained to contain only characters in IA5String.

```
KerberosString  ::= GeneralString (IA5String)
```

In general, US-ASCII control characters should not be used in KerberosString. Control characters SHOULD NOT be used in principal names or realm names.

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 they 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 as 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 encoding change 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.

Future revisions to this protocol will almost certainly allow for a more interoperable representation of principal names, probably including UTF8String.

Note that applying a new constraint to a previously unconstrained type constitutes creation of a new ASN.1 type. In this particular case, the change does not result in a changed encoding under DER.

5.2.2. Realm and PrincipalName

```
Realm           ::= KerberosString

PrincipalName   ::= SEQUENCE {
        name-type       [0] Int32,
        name-string     [1] SEQUENCE OF KerberosString
}
```

Kerberos realm names are encoded as KerberosStrings. Realms shall 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. Acceptable forms for realm names are specified in Section 6.1. A PrincipalName is a typed sequence of components consisting of the following subfields:

name-type
: This field specifies the type of name that follows. Pre-defined values for this field are specified in Section 6.2. 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).

name-string
: This field 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).

5.2.3. KerberosTime

```
KerberosTime    ::= GeneralizedTime -- with no fractional seconds
```

The timestamps used in Kerberos are encoded as GeneralizedTimes. A KerberosTime value shall not include any fractional portions of the seconds. As required by the DER, it further shall not include any separators, and it shall specify 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.

5.2.4. Constrained Integer Types

Some integer members of types SHOULD be constrained to values representable in 32 bits, for compatibility with reasonable implementation limits.

```
Int32           ::= INTEGER (-2147483648..2147483647)
                    -- signed values representable in 32 bits

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

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

Although this results in changes to the abstract types from the RFC 1510 version, the encoding in DER should be unaltered. Historical implementations were typically limited to 32-bit integer values anyway, and assigned numbers SHOULD fall in the space of integer values representable in 32 bits in order to promote interoperability anyway.

Several integer fields in messages are constrained to fixed values.

pvno
: also TKT-VNO or AUTHENTICATOR-VNO, this recurring field is always the constant integer 5. There is no easy way to make this field into a useful protocol version number, so its value is fixed.

msg-type
: this integer field is usually identical to the application tag number of the containing message type.

5.2.5. HostAddress and HostAddresses

```
HostAddress     ::= SEQUENCE  {
        addr-type       [0] Int32,
        address         [1] OCTET STRING
}

-- NOTE: HostAddresses is always used as an OPTIONAL field and
-- should not be empty.
HostAddresses   -- NOTE: subtly different from rfc1510,
                -- but has a value mapping and encodes the same
        ::= SEQUENCE OF HostAddress
```

The host address encodings consist of two fields:

addr-type
: This field specifies the type of address that follows. Pre-defined values for this field are specified in Section 7.5.3.

address
: This field encodes a single address of type addr-type.

5.2.6. AuthorizationData

```
-- NOTE: AuthorizationData is always used as an OPTIONAL field and
-- should not be empty.
AuthorizationData       ::= SEQUENCE OF SEQUENCE {
        ad-type         [0] Int32,
        ad-data         [1] OCTET STRING
}
```

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

ad-type
: This field specifies the format for the ad-data subfield. All negative values are reserved for local use. Non-negative values are reserved for registered use.

Each sequence of type and data 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 elements embedded within them, and the interpretation of the encapsulating element determines which of the embedded elements must be interpreted, and which may be ignored.

These common authorization data elements are recursively defined, meaning that the ad-data for these types will itself contain a sequence of authorization data whose interpretation is affected by the encapsulating element. Depending on the meaning of the encapsulating element, the encapsulated elements may be ignored, might be interpreted as issued directly by the KDC, or might be stored in a separate plaintext part of the ticket. The types of the encapsulating elements are specified as part of the Kerberos specification because the behavior based on these values should be understood across implementations, whereas other elements need only be understood by the applications that they affect.

Authorization data elements are considered critical if present in a ticket or authenticator. If an unknown authorization data element type is received by a server either in an AP-REQ or in a ticket contained in an AP-REQ, then, unless it is encapsulated in a known authorization data element amending the criticality of the elements it contains, authentication MUST 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 the ticket can be used for that service. Authorization elements that are optional can be enclosed in an AD-IF-RELEVANT element.

In the definitions that follow, the value of the ad-type for the element will be specified as the least significant part of the subsection number, and the value of the ad-data will be as shown in the ASN.1 structure that follows the subsection heading.

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

5.2.6.1. IF-RELEVANT

```
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 that may have local extensions for authorization. The ad-type for AD-IF-RELEVANT is (1).

5.2.6.2. KDCIssued

```
AD-KDCIssued            ::= SEQUENCE {
        ad-checksum     [0] Checksum,
        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 that of 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
A sequence of authorization data elements issued by the KDC.

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

The above means 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 when 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). Elements encapsulated with in the KDC-issued element MUST be ignored by the application server if this "signature" is not present. Further, elements encapsulated within this element from a TGT 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 safely be ignored by applications, application servers, and KDCs that do not implement this element.

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

5.2.6.3. AND-OR

```
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).

5.2.6.4. MANDATORY-FOR-KDC

```
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).

5.2.7. PA-DATA

Historically, PA-DATA have been known as "pre-authentication data", meaning that they were used to augment the initial authentication with the KDC. Since that time, they have also been used as a typed hole with which to extend protocol exchanges with the KDC.

```
PA-DATA         ::= SEQUENCE {
        -- NOTE: first tag is [1], not [0]
        padata-type     [1] Int32,
        padata-value    [2] OCTET STRING -- might be encoded AP-REQ
}
```

padata-type
: Indicates the way that the padata-value element is to be interpreted. Negative values of padata-type are reserved for unregistered use; non-negative values are used for a registered interpretation of the element type.

padata-value
: Usually contains the DER encoding of another type; the padata-type field identifies which type is encoded here.

padata-type	Name	Contents of padata-value
1	pa-tgs-req	DER encoding of AP-REQ
2	pa-enc-timestamp	DER encoding of PA-ENC-TIMESTAMP
3	pa-pw-salt	salt (not ASN.1 encoded)
11	pa-etype-info	DER encoding of ETYPE-INFO
19	pa-etype-info2	DER encoding of ETYPE-INFO2

This field MAY also contain information needed by certain extensions to the Kerberos protocol. For example, it might be used to verify the identity of a client initially before any response is returned.

The padata field can also contain information needed to help the KDC or the client select the key needed for generating or decrypting the response. This form of the padata is useful for supporting the use of certain token cards with Kerberos. The details of such extensions are specified in separate documents. See [Pat92] for additional uses of this field.

5.2.7.1. PA-TGS-REQ

In the case of requests for additional tickets (KRB_TGS_REQ), padata-value will contain an encoded AP-REQ. The checksum in the authenticator (which MUST be collision-proof) is to be computed over the KDC-REQ-BODY encoding.

5.2.7.2. Encrypted Timestamp Pre-authentication

There are pre-authentication types that may be used to pre-authenticate a client by means of an encrypted timestamp.

```
PA-ENC-TIMESTAMP        ::= EncryptedData -- PA-ENC-TS-ENC

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

Patimestamp contains the client's time, and pausec contains the microseconds, which MAY be omitted if a client will not generate more than one request per second. The ciphertext (padata-value) consists of the PA-ENC-TS-ENC encoding, encrypted using the client's secret key and a key usage value of 1.

This pre-authentication type was not present in RFC 1510, but many implementations support it.

5.2.7.3. PA-PW-SALT

The padata-value for this pre-authentication type contains the salt for the string-to-key to be used by the client to obtain the key for decrypting the encrypted part of an AS-REP message. Unfortunately, for historical reasons, the character set to be used is unspecified and probably locale-specific.

This pre-authentication type was not present in RFC 1510, but many implementations support it. It is necessary in any case where the salt for the string-to-key algorithm is not the default.

In the trivial example, a zero-length salt string is very commonplace for realms that have converted their principal databases from Kerberos Version 4.

A KDC SHOULD NOT send PA-PW-SALT when issuing a KRB-ERROR message that requests additional pre-authentication. Implementation note: Some KDC implementations issue an erroneous PA-PW-SALT when issuing a KRB-ERROR message that requests additional pre-authentication. Therefore, clients SHOULD ignore a PA-PW-SALT accompanying a KRB-ERROR message that requests additional pre-authentication. As noted in section 3.1.3, a KDC MUST NOT send PA-PW-SALT when the client's AS-REQ includes at least one "newer" etype.

5.2.7.4. PA-ETYPE-INFO

The ETYPE-INFO pre-authentication type is sent by the KDC in a KRB-ERROR indicating a requirement for additional pre-authentication. It is usually used to notify a client of which key to use for the encryption of an encrypted timestamp for the purposes of sending a PA-ENC-TIMESTAMP pre-authentication value. It MAY also be sent in an AS-REP to provide information to the client about which key salt to use for the string-to-key to be used by the client to obtain the key for decrypting the encrypted part the AS-REP.

```
ETYPE-INFO-ENTRY        ::= SEQUENCE {
        etype           [0] Int32,
        salt            [1] OCTET STRING OPTIONAL
}

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

The salt, like that of PA-PW-SALT, is also completely unspecified with respect to character set and is probably locale-specific.

If ETYPE-INFO is sent in an AS-REP, there shall be exactly one ETYPE-INFO-ENTRY, and its etype shall match that of the enc-part in the AS-REP.

This pre-authentication type was not present in RFC 1510, but many implementations that support encrypted timestamps for pre-authentication need to support ETYPE-INFO as well. As noted in Section 3.1.3, a KDC MUST NOT send PA-ETYPE-INFO when the client's AS-REQ includes at least one "newer" etype.

5.2.7.5. PA-ETYPE-INFO2

The ETYPE-INFO2 pre-authentication type is sent by the KDC in a KRB-ERROR indicating a requirement for additional pre-authentication. It is usually used to notify a client of which key to use for the encryption of an encrypted timestamp for the purposes of sending a PA-ENC-TIMESTAMP pre-authentication value. It MAY also be sent in an AS-REP to provide information to the client about which key salt to use for the string-to-key to be used by the client to obtain the key for decrypting the encrypted part the AS-REP.

```
ETYPE-INFO2-ENTRY       ::= SEQUENCE {
        etype           [0] Int32,
        salt            [1] KerberosString OPTIONAL,
        s2kparams       [2] OCTET STRING OPTIONAL
}

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

The type of the salt is KerberosString, but existing installations might have locale-specific characters stored in salt strings, and implementors MAY choose to handle them.

The interpretation of s2kparams is specified in the cryptosystem description associated with the etype. Each cryptosystem has a default interpretation of s2kparams that will hold if that element is omitted from the encoding of ETYPE-INFO2-ENTRY.

If ETYPE-INFO2 is sent in an AS-REP, there shall be exactly one ETYPE-INFO2-ENTRY, and its etype shall match that of the enc-part in the AS-REP.

The preferred ordering of the "hint" pre-authentication data that affect client key selection is: ETYPE-INFO2, followed by ETYPE-INFO, followed by PW-SALT. As noted in Section 3.1.3, a KDC MUST NOT send ETYPE-INFO or PW-SALT when the client's AS-REQ includes at least one "newer" etype.

The ETYPE-INFO2 pre-authentication type was not present in RFC 1510.

5.2.8. KerberosFlags

For several message types, a specific constrained bit string type, KerberosFlags, is used.

```
KerberosFlags   ::= BIT STRING (SIZE (32..MAX))
                    -- minimum number of bits shall be sent,
                    -- but no fewer than 32
```

Compatibility note: The following paragraphs describe a change from the RFC 1510 description of bit strings that would result in incompatility in the case of an implementation that strictly conformed to ASN.1 DER and RFC 1510.

ASN.1 bit strings have multiple uses. The simplest use of a bit string is to contain a vector of bits, with no particular meaning attached to individual bits. This vector of bits is not necessarily a multiple of eight bits long. The use in Kerberos of a bit string as a compact boolean vector wherein each element has a distinct meaning poses some problems. The natural notation for a compact boolean vector is the ASN.1 "NamedBit" notation, and the DER require that encodings of a bit string using "NamedBit" notation exclude any trailing zero bits. This truncation is easy to neglect, especially given C language implementations that naturally choose to store boolean vectors as 32-bit integers.

For example, if the notation for KDCOptions were to include the "NamedBit" notation, as in RFC 1510, and a KDCOptions value to be encoded had only the "forwardable" (bit number one) bit set, the DER encoding MUST include only two bits: the first reserved bit ("reserved", bit number zero, value zero) and the one-valued bit (bit number one) for "forwardable".

Most existing implementations of Kerberos unconditionally send 32 bits on the wire when encoding bit strings used as boolean vectors. This behavior violates the ASN.1 syntax used for flag values in RFC 1510, but it occurs on such a widely installed base that the protocol description is being modified to accommodate it.

Consequently, this document removes the "NamedBit" notations for individual bits, relegating them to comments. The size constraint on the KerberosFlags type requires that at least 32 bits be encoded at all times, though a lenient implementation MAY choose to accept fewer than 32 bits and to treat the missing bits as set to zero.

Currently, no uses of KerberosFlags specify more than 32 bits' worth of flags, although future revisions of this document may do so. When more than 32 bits are to be transmitted in a KerberosFlags value, future revisions to this document will likely specify that the smallest number of bits needed to encode the highest-numbered one-valued bit should be sent. This is somewhat similar to the DER encoding of a bit string that is declared with the "NamedBit" notation.

5.2.9. Cryptosystem-Related Types

Many Kerberos protocol messages contain an EncryptedData as a container for arbitrary encrypted data, which is often the encrypted encoding of another data type. Fields within EncryptedData assist the recipient in selecting a key with which to decrypt the enclosed data.

```
EncryptedData   ::= SEQUENCE {
        etype   [0] Int32 -- EncryptionType --,
        kvno    [1] UInt32 OPTIONAL,
        cipher  [2] OCTET STRING -- ciphertext
}
```

etype
: This field identifies which encryption algorithm was used to encipher the cipher.

kvno
: This field contains the version number of the key under which data is encrypted. It is only present in messages encrypted under long lasting keys, such as principals' secret keys.

cipher
: This field contains the enciphered text, encoded as an OCTET STRING. (Note that the encryption mechanisms defined in [RFC3961] MUST incorporate integrity protection as well, so no additional checksum is required.)

The EncryptionKey type is the means by which cryptographic keys used for encryption are transferred.

```
EncryptionKey   ::= SEQUENCE {
        keytype         [0] Int32 -- actually encryption type --,
        keyvalue        [1] OCTET STRING
}
```

keytype
This field specifies the encryption type of the encryption key that follows in the keyvalue field. Although its name is "keytype", it actually specifies an encryption type. Previously, multiple cryptosystems that performed encryption differently but were capable of using keys with the same characteristics were permitted to share an assigned number to designate the type of key; this usage is now deprecated.

keyvalue
This field contains the key itself, encoded as an octet string.

Messages containing cleartext data to be authenticated will usually do so by using a member of type Checksum. Most instances of Checksum use a keyed hash, though exceptions will be noted.

```
Checksum        ::= SEQUENCE {
        cksumtype       [0] Int32,
        checksum        [1] OCTET STRING
}
```

cksumtype
This field indicates the algorithm used to generate the accompanying checksum.

checksum
This field contains the checksum itself, encoded as an octet string.

See Section 4 for a brief description of the use of encryption and checksums in Kerberos.

5.3. Tickets

This section describes the format and encryption parameters for tickets and authenticators. When a ticket or authenticator is included in a protocol message, it is treated as an opaque object. A ticket is a record that helps a client authenticate to a service. A Ticket contains the following information:

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

-- Encrypted part of ticket
```

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

   -- encoded Transited field
   TransitedEncoding       ::= SEQUENCE {
           tr-type         [0] Int32 -- must be registered --,
           contents        [1] OCTET STRING
   }

   TicketFlags     ::= KerberosFlags
           -- 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),
   -- the following are new since 1510
           -- transited-policy-checked(12),
           -- ok-as-delegate(13)
```

tkt-vno
   This field specifies the version number for the ticket format. This document describes version number 5.

realm
   This field specifies the realm that issued a ticket. It also serves to identify the realm part of the server's principal identifier. Since a Kerberos server can only issue tickets for servers within its realm, the two will always be identical.

sname
This field specifies all components of the name part of the server's identity, including those parts that identify a specific instance of a service.

enc-part
This field holds the encrypted encoding of the EncTicketPart sequence. It is encrypted in the key shared by Kerberos and the end server (the server's secret key), using a key usage value of 2.

flags
This field indicates which of various options were used or requested when the ticket was issued. The meanings of the flags are as follows:

Bit(s)	Name	Description
0	reserved	Reserved for future expansion of this field.
1	forwardable	The FORWARDABLE flag is normally only interpreted by the TGS, and can be ignored by end servers. When set, this flag tells the ticket-granting server that it is OK to issue a new TGT with a different network address based on the presented ticket.
2	forwarded	When set, this flag indicates that the ticket has either been forwarded or was issued based on authentication involving a forwarded TGT.
3	proxiable	The PROXIABLE flag is normally only interpreted by the TGS, and can be ignored by end servers. The PROXIABLE flag has an interpretation identical to that of the FORWARDABLE flag, except that the PROXIABLE flag tells the ticket-granting server that only non-TGTs may be issued with different network addresses.
4	proxy	When set, this flag indicates that a ticket is a proxy.
5	may-postdate	The MAY-POSTDATE flag is normally only interpreted by the TGS, and can be ignored by end servers. This flag tells the

		ticket-granting server that a post-dated ticket MAY be issued based on this TGT.
6	postdated	This flag indicates that this ticket has been postdated. The end-service can check the authtime field to see when the original authentication occurred.
7	invalid	This flag indicates that a ticket is invalid, and it must be validated by the KDC before use. Application servers must reject tickets which have this flag set.
8	renewable	The RENEWABLE flag is normally only interpreted by the TGS, and can usually be ignored by end servers (some particularly careful servers MAY disallow renewable tickets). A renewable ticket can be used to obtain a replacement ticket that expires at a later date.
9	initial	This flag indicates that this ticket was issued using the AS protocol, and not issued based on a TGT.
10	pre-authent	This flag indicates that during initial authentication, the client was authenticated by the KDC before a ticket was issued. The strength of the pre-authentication method is not indicated, but is acceptable to the KDC.
11	hw-authent	This flag indicates that the protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client. The hardware authentication method is selected by the KDC and the strength of the method is not indicated.
12	transited-policy-checked	This flag indicates that the KDC for the realm has checked the transited field against a realm-defined policy for trusted certifiers. If this flag is reset (0), then the application server must check the transited field itself, and if unable to do so, it must reject the authentication. If the flag is set (1), then the application server MAY skip its own validation of the

		transited field, relying on the validation performed by the KDC. At its option the application server MAY still apply its own validation based on a separate policy for acceptance. This flag is new since RFC 1510.
13	ok-as-delegate	This flag indicates that the server (not the client) 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 decide whether to delegate credentials (either grant a proxy or a forwarded TGT) to this server. The client is free 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. This flag is new since RFC 1510.
14-31	reserved	Reserved for future use.

key
: This field exists in the ticket and the KDC response and is used to pass the session key from Kerberos to the application server and the client.

crealm
: This field contains the name of the realm in which the client is registered and in which initial authentication took place.

cname
: This field contains the name part of the client's principal identifier.

transited
: This field lists the names of the Kerberos realms that took part in authenticating the user to whom this ticket was issued. It does not specify the order in which the realms were transited. See Section 3.3.3.2 for details on how this field encodes the traversed realms. When the names of CAs are to be embedded in the transited field (as specified for some extensions to the

protocol), the X.500 names of the CAs SHOULD be mapped into items in the transited field using the mapping defined by RFC 2253.

authtime
: This field indicates the time of initial authentication for the named principal. It is the time of issue for the original ticket on which this ticket is based. It is included in the ticket to provide additional information to the end service, and to provide the necessary information for implementation of a "hot list" service at the KDC. An end service that is particularly paranoid could refuse to accept tickets for which the initial authentication occurred "too far" in the past. This field is also returned as part of the response from the KDC. When it is returned as part of the response to initial authentication (KRB_AS_REP), this is the current time on the Kerberos server. It is NOT recommended that this time value be used to adjust the workstation's clock, as the workstation cannot reliably determine that such a KRB_AS_REP actually came from the proper KDC in a timely manner.

starttime
: This field in the ticket specifies the time after which the ticket is valid. Together with endtime, this field specifies the life of the ticket. If the starttime field is absent from the ticket, then the authtime field SHOULD be used in its place to determine the life of the ticket.

endtime
: This field 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 field 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.

caddr
: This field in a ticket contains zero (if omitted) or more (if present) host addresses. These are the addresses from which the ticket can be used. If there are no addresses, the ticket can be used from any location. The decision by the KDC to issue or by the end server to accept addressless tickets is a policy decision and is left to the Kerberos and end-service administrators; they MAY refuse to issue or accept such tickets. Because of the wide

deployment of network address translation, it is recommended that policy allow the issue and acceptance of such tickets.

Network addresses are included in the ticket to make it harder for an attacker to use stolen credentials. Because the session key is not sent over the network in cleartext, credentials can't be stolen simply by listening to the network; an attacker has to gain access to the session key (perhaps through operating system security breaches or a careless user's unattended session) to make use of stolen tickets.

Note that the network address from which a connection is received cannot be reliably determined. Even if it could be, an attacker who has compromised the client's workstation could use the credentials from there. Including the network addresses only makes it more difficult, not impossible, for an attacker to walk off with stolen credentials and then to use them from a "safe" location.

authorization-data

The authorization-data field is used to pass authorization data from the principal on whose behalf a ticket was issued to the application service. If no authorization data is included, this field will be left out. Experience has shown that the name of this field is confusing, and that a better name would be "restrictions". Unfortunately, it is not possible to change the name at this time.

This field contains restrictions on any authority obtained on the basis of authentication using the ticket. It is possible for any principal in possession of credentials to add entries to the authorization data field since these entries further restrict what can be done with the ticket. Such additions can be made by specifying the additional entries when a new ticket is obtained during the TGS exchange, or they MAY be added during chained delegation using the authorization data field of the authenticator.

Because entries may be added to this field by the holder of credentials, except when an entry is separately authenticated by encapsulation in the KDC-issued element, it is not allowable for the presence of an entry in the authorization data field of a ticket to amplify the privileges one would obtain from using a ticket.

The data in this field may be specific to the end service; the field will contain the names of service specific objects, and the rights to those objects. The format for this field is described

in Section 5.2.6. Although Kerberos is not concerned with the format of the contents of the subfields, it does carry type information (ad-type).

By using the authorization_data field, a principal is able to issue a proxy that is valid for a specific purpose. For example, a client wishing to print a file can obtain a file server proxy to be passed to the print server. By specifying the name of the file in the authorization_data field, the file server knows that the print server can only use the client's rights when accessing the particular file to be printed.

A separate service providing authorization or certifying group membership may be built using the authorization-data field. In this case, the entity granting authorization (not the authorized entity) may obtain a ticket in its own name (e.g., the ticket is issued in the name of a privilege server), and this entity adds restrictions on its own authority and delegates the restricted authority through a proxy to the client. The client would then present this authorization credential to the application server separately from the authentication exchange. Alternatively, such authorization credentials MAY be embedded in the ticket authenticating the authorized entity, when the authorization is separately authenticated using the KDC-issued authorization data element (see 5.2.6.2).

Similarly, if one specifies the authorization-data field of a proxy and leaves the host addresses blank, the resulting ticket and session key can be treated as a capability. See [Neu93] for some suggested uses of this field.

The authorization-data field is optional and does not have to be included in a ticket.

5.4. Specifications for the AS and TGS Exchanges

This section specifies the format of the messages used in the exchange between the client and the Kerberos server. The format of possible error messages appears in Section 5.9.1.

5.4.1. KRB_KDC_REQ Definition

The KRB_KDC_REQ message has no application tag number of its own. Instead, it is incorporated into either KRB_AS_REQ or KRB_TGS_REQ, each of which has an application tag, depending on whether the request is for an initial ticket or an additional ticket. In either case, the message is sent from the client to the KDC to request credentials for a service.

The message fields are as follows:

```
AS-REQ          ::= [APPLICATION 10] KDC-REQ

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

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

KDC-REQ-BODY    ::= 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,
        rtime                   [6] KerberosTime OPTIONAL,
        nonce                   [7] UInt32,
        etype                   [8] SEQUENCE OF Int32 -- EncryptionType
                                    -- in preference order --,
        addresses               [9] HostAddresses OPTIONAL,
        enc-authorization-data  [10] EncryptedData OPTIONAL
                                    -- AuthorizationData --,
        additional-tickets      [11] SEQUENCE OF Ticket OPTIONAL
                                       -- NOTE: not empty
}

KDCOptions      ::= KerberosFlags
        -- reserved(0),
        -- forwardable(1),
        -- forwarded(2),
        -- proxiable(3),
        -- proxy(4),
        -- allow-postdate(5),
        -- postdated(6),
        -- unused7(7),
        -- renewable(8),
        -- unused9(9),
        -- unused10(10),
```

```
        -- opt-hardware-auth(11),
        -- unused12(12),
        -- unused13(13),
-- 15 is reserved for canonicalize
        -- unused15(15),
-- 26 was unused in 1510
        -- disable-transited-check(26),
--
        -- renewable-ok(27),
        -- enc-tkt-in-skey(28),
        -- renew(30),
        -- validate(31)
```

The fields in this message are as follows:

pvno
: This field is included in each message, and specifies the protocol version number. This document specifies protocol version 5.

msg-type
: This field indicates the type of a protocol message. It will almost always be the same as the application identifier associated with a message. It is included to make the identifier more readily accessible to the application. For the KDC-REQ message, this type will be KRB_AS_REQ or KRB_TGS_REQ.

padata
: Contains pre-authentication data. Requests for additional tickets (KRB_TGS_REQ) MUST contain a padata of PA-TGS-REQ.

  The padata (pre-authentication data) field contains a sequence of authentication information that may be needed before credentials can be issued or decrypted.

req-body
: This field is a placeholder delimiting the extent of the remaining fields. If a checksum is to be calculated over the request, it is calculated over an encoding of the KDC-REQ-BODY sequence which is enclosed within the req-body field.

kdc-options
: This field appears in the KRB_AS_REQ and KRB_TGS_REQ requests to the KDC and indicates the flags that the client wants set on the tickets as well as other information that is to modify the behavior of the KDC. Where appropriate, the name of an option may be the same as the flag that is set by that option. Although in most cases, the bit in the options field will be the same as that in the flags field, this is not guaranteed, so it is not

acceptable simply to copy the options field to the flags field. There are various checks that must be made before an option is honored anyway.

The kdc_options field is a bit-field, where the selected options are indicated by the bit being set (1), and the unselected options and reserved fields being reset (0). The encoding of the bits is specified in Section 5.2. The options are described in more detail above in Section 2. The meanings of the options are as follows:

Bits	Name	Description
0	RESERVED	Reserved for future expansion of this field.
1	FORWARDABLE	The FORWARDABLE option indicates that the ticket to be issued is to have its forwardable flag set. It may only be set on the initial request, or in a subsequent request if the TGT on which it is based is also forwardable.
2	FORWARDED	The FORWARDED option is only specified in a request to the ticket-granting server and will only be honored if the TGT in the request has its FORWARDABLE bit set. This option indicates that this is a request for forwarding. The address(es) of the host from which the resulting ticket is to be valid are included in the addresses field of the request.
3	PROXIABLE	The PROXIABLE option indicates that the ticket to be issued is to have its proxiable flag set. It may only be set on the initial request, or a subsequent request if the TGT on which it is based is also proxiable.
4	PROXY	The PROXY option indicates that this is a request for a proxy. This option will only be honored if the TGT in the request has its PROXIABLE bit set. The address(es) of the

		host from which the resulting ticket is to be valid are included in the addresses field of the request.
5	ALLOW-POSTDATE	The ALLOW-POSTDATE option indicates that the ticket to be issued is to have its MAY-POSTDATE flag set. It may only be set on the initial request, or in a subsequent request if the TGT on which it is based also has its MAY-POSTDATE flag set.
6	POSTDATED	The POSTDATED option indicates that this is a request for a postdated ticket. This option will only be honored if the TGT on which it is based has its MAY-POSTDATE flag set. The resulting ticket will also have its INVALID flag set, and that flag may be reset by a subsequent request to the KDC after the starttime in the ticket has been reached.
7	RESERVED	This option is presently unused.
8	RENEWABLE	The RENEWABLE option indicates that the ticket to be issued is to have its RENEWABLE flag set. It may only be set on the initial request, or when the TGT on which the request is based is also renewable. If this option is requested, then the rtime field in the request contains the desired absolute expiration time for the ticket.
9	RESERVED	Reserved for PK-Cross.
10	RESERVED	Reserved for future use.
11	RESERVED	Reserved for opt-hardware-auth.
12-25	RESERVED	Reserved for future use.
26	DISABLE-TRANSITED-CHECK	By default the KDC will check the transited field of a TGT against the policy of the local realm before it will issue derivative tickets based

		on the TGT. If this flag is set in the request, checking of the transited field is disabled. Tickets issued without the performance of this check will be noted by the reset (0) value of the TRANSITED-POLICY-CHECKED flag, indicating to the application server that the transited field must be checked locally. KDCs are encouraged but not required to honor the DISABLE-TRANSITED-CHECK option. This flag is new since RFC 1510.
27	RENEWABLE-OK	The RENEWABLE-OK option indicates that a renewable ticket will be acceptable if a ticket with the requested life cannot otherwise be provided, in which case a renewable ticket may be issued with a renew-till equal to the requested endtime. The value of the renew-till field may still be limited by local limits, or limits selected by the individual principal or server.
28	ENC-TKT-IN-SKEY	This option is used only by the ticket-granting service. The ENC-TKT-IN-SKEY option indicates that the ticket for the end server is to be encrypted in the session key from the additional TGT provided.
29	RESERVED	Reserved for future use.
30	RENEW	This option is used only by the ticket-granting service. The RENEW option indicates that the present request is for a renewal. The ticket provided is encrypted in the secret key for the server on which it is valid. This option will only be honored if the ticket to be renewed has its RENEWABLE flag set and if the time in its renew-till field has not passed. The ticket to be renewed is passed in the padata

		field as part of the authentication header.
31	VALIDATE	This option is used only by the ticket-granting service. The VALIDATE option indicates that the request is to validate a postdated ticket. It will only be honored if the ticket presented is postdated, presently has its INVALID flag set, and would otherwise be usable at this time. A ticket cannot be validated before its starttime. The ticket presented for validation is encrypted in the key of the server for which it is valid and is passed in the padata field as part of the authentication header.

cname and sname
: These fields are the same as those described for the ticket in section 5.3. The sname may only be absent when the ENC-TKT-IN-SKEY option is specified. If the sname is absent, the name of the server is taken from the name of the client in the ticket passed as additional-tickets.

enc-authorization-data
: The enc-authorization-data, if present (and it can only be present in the TGS_REQ form), is an encoding of the desired authorization-data encrypted under the sub-session key if present in the Authenticator, or alternatively from the session key in the TGT (both the Authenticator and TGT come from the padata field in the KRB_TGS_REQ). The key usage value used when encrypting is 5 if a sub-session key is used, or 4 if the session key is used.

realm
: This field specifies the realm part of the server's principal identifier. In the AS exchange, this is also the realm part of the client's principal identifier.

from
: This field is included in the KRB_AS_REQ and KRB_TGS_REQ ticket requests when the requested ticket is to be postdated. It specifies the desired starttime for the requested ticket. If this field is omitted, then the KDC SHOULD use the current time instead.

till
: This field contains the expiration date requested by the client in a ticket request. It is not optional, but if the requested endtime is "19700101000000Z", the requested ticket is to have the maximum endtime permitted according to KDC policy. Implementation note: This special timestamp corresponds to a UNIX time_t value of zero on most systems.

rtime
: This field is the requested renew-till time sent from a client to the KDC in a ticket request. It is optional.

nonce
: This field is part of the KDC request and response. It is intended to hold a random number generated by the client. If the same number is included in the encrypted response from the KDC, it provides evidence that the response is fresh and has not been replayed by an attacker. Nonces MUST NEVER be reused.

etype
: This field specifies the desired encryption algorithm to be used in the response.

addresses
: This field is included in the initial request for tickets, and it is optionally included in requests for additional tickets from the ticket-granting server. It specifies the addresses from which the requested ticket is to be valid. Normally it includes the addresses for the client's host. If a proxy is requested, this field will contain other addresses. The contents of this field are usually copied by the KDC into the caddr field of the resulting ticket.

additional-tickets
: Additional tickets MAY be optionally included in a request to the ticket-granting server. If the ENC-TKT-IN-SKEY option has been specified, then the session key from the additional ticket will be used in place of the server's key to encrypt the new ticket. When the ENC-TKT-IN-SKEY option is used for user-to-user authentication, this additional ticket MAY be a TGT issued by the local realm or an inter-realm TGT issued for the current KDC's realm by a remote KDC. If more than one option that requires additional tickets has been specified, then the additional tickets are used in the order specified by the ordering of the options bits (see kdc-options, above).

The application tag number will be either ten (10) or twelve (12) depending on whether the request is for an initial ticket (AS-REQ) or for an additional ticket (TGS-REQ).

The optional fields (addresses, authorization-data, and additional-tickets) are only included if necessary to perform the operation specified in the kdc-options field.

Note that in KRB_TGS_REQ, the protocol version number appears twice and two different message types appear: the KRB_TGS_REQ message contains these fields as does the authentication header (KRB_AP_REQ) that is passed in the padata field.

5.4.2. KRB_KDC_REP Definition

The KRB_KDC_REP message format is used for the reply from the KDC for either an initial (AS) request or a subsequent (TGS) request. There is no message type for KRB_KDC_REP. Instead, the type will be either KRB_AS_REP or KRB_TGS_REP. The key used to encrypt the ciphertext part of the reply depends on the message type. For KRB_AS_REP, the ciphertext is encrypted in the client's secret key, and the client's key version number is included in the key version number for the encrypted data. For KRB_TGS_REP, the ciphertext is encrypted in the sub-session key from the Authenticator; if it is absent, the ciphertext is encrypted in the session key from the TGT used in the request. In that case, no version number will be present in the EncryptedData sequence.

The KRB_KDC_REP message contains the following fields:

```
AS-REP          ::= [APPLICATION 11] KDC-REP

TGS-REP         ::= [APPLICATION 13] KDC-REP

KDC-REP         ::= SEQUENCE {
        pvno            [0] INTEGER (5),
        msg-type        [1] INTEGER (11 -- AS -- | 13 -- TGS --),
        padata          [2] SEQUENCE OF PA-DATA OPTIONAL
                                -- NOTE: not empty --,
        crealm          [3] Realm,
        cname           [4] PrincipalName,
        ticket          [5] Ticket,
        enc-part        [6] EncryptedData
                                -- EncASRepPart or EncTGSRepPart,
                                -- as appropriate
}

EncASRepPart    ::= [APPLICATION 25] EncKDCRepPart
```

```
EncTGSRepPart   ::= [APPLICATION 26] EncKDCRepPart

EncKDCRepPart   ::= SEQUENCE {
        key             [0] EncryptionKey,
        last-req        [1] LastReq,
        nonce           [2] UInt32,
        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
}

LastReq         ::=     SEQUENCE OF SEQUENCE {
        lr-type         [0] Int32,
        lr-value        [1] KerberosTime
}
```

pvno and msg-type
: These fields are described above in Section 5.4.1. msg-type is either KRB_AS_REP or KRB_TGS_REP.

padata
: This field is described in detail in Section 5.4.1. One possible use for it is to encode an alternate "salt" string to be used with a string-to-key algorithm. This ability is useful for easing transitions if a realm name needs to change (e.g., when a company is acquired); in such a case all existing password-derived entries in the KDC database would be flagged as needing a special salt string until the next password change.

crealm, cname, srealm, and sname
: These fields are the same as those described for the ticket in section 5.3.

ticket
: The newly-issued ticket, from Section 5.3.

enc-part
: This field is a place holder for the ciphertext and related information that forms the encrypted part of a message. The description of the encrypted part of the message follows each appearance of this field.

The key usage value for encrypting this field is 3 in an AS-REP message, using the client's long-term key or another key selected via pre-authentication mechanisms. In a TGS-REP message, the key usage value is 8 if the TGS session key is used, or 9 if a TGS authenticator subkey is used.

Compatibility note: Some implementations unconditionally send an encrypted EncTGSRepPart (application tag number 26) in this field regardless of whether the reply is a AS-REP or a TGS-REP. In the interest of compatibility, implementors MAY relax the check on the tag number of the decrypted ENC-PART.

key
This field is the same as described for the ticket in Section 5.3.

last-req
This field is returned by the KDC and specifies the time(s) of the last request by a principal. Depending on what information is available, this might be the last time that a request for a TGT was made, or the last time that a request based on a TGT was successful. It also might cover all servers for a realm, or just the particular server. Some implementations MAY display this information to the user to aid in discovering unauthorized use of one's identity. It is similar in spirit to the last login time displayed when logging in to timesharing systems.

lr-type
This field indicates how the following lr-value field is to be interpreted. Negative values indicate that the information pertains only to the responding server. Non-negative values pertain to all servers for the realm.

If the lr-type field is zero (0), then no information is conveyed by the lr-value subfield. If the absolute value of the lr-type field is one (1), then the lr-value subfield is the time of last initial request for a TGT. If it is two (2), then the lr-value subfield is the time of last initial request. If it is three (3), then the lr-value subfield is the time of issue for the newest TGT used. If it is four (4), then the lr-value subfield is the time of the last renewal. If it is five (5), then the lr-value subfield is the time of last request (of any type). If it is (6), then the lr-value subfield is the time when the password will expire. If it is (7), then the lr-value subfield is the time when the account will expire.

lr-value
This field contains the time of the last request. The time MUST be interpreted according to the contents of the accompanying lr-type subfield.

nonce
This field is described above in Section 5.4.1.

key-expiration
The key-expiration field is part of the response from the KDC and specifies the time that the client's secret key is due to expire. The expiration might be the result of password aging or an account expiration. If present, it SHOULD be set to the earlier of the user's key expiration and account expiration. The use of this field is deprecated, and the last-req field SHOULD be used to convey this information instead. This field will usually be left out of the TGS reply since the response to the TGS request is encrypted in a session key and no client information has to be retrieved from the KDC database. It is up to the application client (usually the login program) to take appropriate action (such as notifying the user) if the expiration time is imminent.

flags, authtime, starttime, endtime, renew-till and caddr
These fields are duplicates of those found in the encrypted portion of the attached ticket (see Section 5.3), provided so the client MAY verify that they match the intended request and in order to assist in proper ticket caching. If the message is of type KRB_TGS_REP, the caddr field will only be filled in if the request was for a proxy or forwarded ticket, or if the user is substituting a subset of the addresses from the TGT. If the client-requested addresses are not present or not used, then the addresses contained in the ticket will be the same as those included in the TGT.

5.5. Client/Server (CS) Message Specifications

This section specifies the format of the messages used for the authentication of the client to the application server.

5.5.1. KRB_AP_REQ Definition

The KRB_AP_REQ message contains the Kerberos protocol version number, the message type KRB_AP_REQ, an options field to indicate any options in use, and the ticket and authenticator themselves. The KRB_AP_REQ message is often referred to as the "authentication header".

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

APOptions       ::= KerberosFlags
        -- reserved(0),
        -- use-session-key(1),
        -- mutual-required(2)
```

pvno and msg-type

These fields are described above in Section 5.4.1. msg-type is KRB_AP_REQ.

ap-options

This field appears in the application request (KRB_AP_REQ) and affects the way the request is processed. It is a bit-field, where the selected options are indicated by the bit being set (1), and the unselected options and reserved fields by being reset (0). The encoding of the bits is specified in Section 5.2. The meanings of the options are as follows:

Bit(s)	Name	Description
0	reserved	Reserved for future expansion of this field.
1	use-session-key	The USE-SESSION-KEY option indicates that the ticket the client is presenting to a server is encrypted in the session key from the server's TGT. When this option is not specified, the ticket is encrypted in the server's secret key.
2	mutual-required	The MUTUAL-REQUIRED option tells the server that the client requires mutual authentication, and that it must respond with a KRB_AP_REP message.
3-31	reserved	Reserved for future use.

ticket

This field is a ticket authenticating the client to the server.

authenticator
This contains the encrypted authenticator, which includes the client's choice of a subkey.

The encrypted authenticator is included in the AP-REQ; it certifies to a server that the sender has recent knowledge of the encryption key in the accompanying ticket, to help the server detect replays. It also assists in the selection of a "true session key" to use with the particular session. The DER encoding of the following is encrypted in the ticket's session key, with a key usage value of 11 in normal application exchanges, or 7 when used as the PA-TGS-REQ PA-DATA field of a TGS-REQ exchange (see Section 5.4.1):

```
-- Unencrypted authenticator
Authenticator   ::= [APPLICATION 2] SEQUENCE  {
        authenticator-vno       [0] INTEGER (5),
        crealm                  [1] Realm,
        cname                   [2] PrincipalName,
        cksum                   [3] Checksum OPTIONAL,
        cusec                   [4] Microseconds,
        ctime                   [5] KerberosTime,
        subkey                  [6] EncryptionKey OPTIONAL,
        seq-number              [7] UInt32 OPTIONAL,
        authorization-data      [8] AuthorizationData OPTIONAL
}
```

authenticator-vno
This field specifies the version number for the format of the authenticator. This document specifies version 5.

crealm and cname
These fields are the same as those described for the ticket in section 5.3.

cksum
This field contains a checksum of the application data that accompanies the KRB_AP_REQ, computed using a key usage value of 10 in normal application exchanges, or 6 when used in the TGS-REQ PA-TGS-REQ AP-DATA field.

cusec
This field contains the microsecond part of the client's timestamp. Its value (before encryption) ranges from 0 to 999999. It often appears along with ctime. The two fields are used together to specify a reasonably accurate timestamp.

ctime
This field contains the current time on the client's host.

subkey
This field contains the client's choice for an encryption key to be used to protect this specific application session. Unless an application specifies otherwise, if this field is left out, the session key from the ticket will be used.

seq-number
This optional field includes the initial sequence number to be used by the KRB_PRIV or KRB_SAFE messages when sequence numbers are used to detect replays. (It may also be used by application specific messages.) When included in the authenticator, this field specifies the initial sequence number for messages from the client to the server. When included in the AP-REP message, the initial sequence number is that for messages from the server to the client. When used in KRB_PRIV or KRB_SAFE messages, it is incremented by one after each message is sent. Sequence numbers fall in the range 0 through 2^32 - 1 and wrap to zero following the value 2^32 - 1.

For sequence numbers to support the detection of replays adequately, they SHOULD be non-repeating, even across connection boundaries. The initial sequence number SHOULD be random and uniformly distributed across the full space of possible sequence numbers, so that it cannot be guessed by an attacker and so that it and the successive sequence numbers do not repeat other sequences. In the event that more than 2^32 messages are to be generated in a series of KRB_PRIV or KRB_SAFE messages, rekeying SHOULD be performed before sequence numbers are reused with the same encryption key.

Implmentation note: Historically, some implementations transmit signed twos-complement numbers for sequence numbers. In the interests of compatibility, implementations MAY accept the equivalent negative number where a positive number greater than 2^31 - 1 is expected.

Implementation note: As noted before, some implementations omit the optional sequence number when its value would be zero. Implementations MAY accept an omitted sequence number when expecting a value of zero, and SHOULD NOT transmit an Authenticator with a initial sequence number of zero.

authorization-data
This field is the same as described for the ticket in Section 5.3. It is optional and will only appear when additional restrictions are to be placed on the use of a ticket, beyond those carried in the ticket itself.

5.5.2. KRB_AP_REP Definition

The KRB_AP_REP message contains the Kerberos protocol version number, the message type, and an encrypted time-stamp. The message is sent in response to an application request (KRB_AP_REQ) for which the mutual authentication option has been selected in the ap-options field.

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

EncAPRepPart    ::= [APPLICATION 27] SEQUENCE {
        ctime           [0] KerberosTime,
        cusec           [1] Microseconds,
        subkey          [2] EncryptionKey OPTIONAL,
        seq-number      [3] UInt32 OPTIONAL
}
```

The encoded EncAPRepPart is encrypted in the shared session key of the ticket. The optional subkey field can be used in an application-arranged negotiation to choose a per association session key.

pvno and msg-type
: These fields are described above in Section 5.4.1. msg-type is KRB_AP_REP.

enc-part
: This field is described above in Section 5.4.2. It is computed with a key usage value of 12.

ctime
: This field contains the current time on the client's host.

cusec
: This field contains the microsecond part of the client's timestamp.

subkey
: This field contains an encryption key that is to be used to protect this specific application session. See Section 3.2.6 for specifics on how this field is used to negotiate a key. Unless an application specifies otherwise, if this field is left out, the sub-session key from the authenticator or if the latter is also left out, the session key from the ticket will be used.

seq-number
This field is described above in Section 5.3.2.

5.5.3. Error Message Reply

If an error occurs while processing the application request, the KRB_ERROR message will be sent in response. See Section 5.9.1 for the format of the error message. The cname and crealm fields MAY be left out if the server cannot determine their appropriate values from the corresponding KRB_AP_REQ message. If the authenticator was decipherable, the ctime and cusec fields will contain the values from it.

5.6. KRB_SAFE Message Specification

This section specifies the format of a message that can be used by either side (client or server) of an application to send a tamper-proof message to its peer. It presumes that a session key has previously been exchanged (for example, by using the KRB_AP_REQ/KRB_AP_REP messages).

5.6.1. KRB_SAFE definition

The KRB_SAFE message contains user data along with a collision-proof checksum keyed with the last encryption key negotiated via subkeys, or with the session key if no negotiation has occurred. The message fields are as follows:

```
KRB-SAFE        ::= [APPLICATION 20] SEQUENCE {
        pvno            [0] INTEGER (5),
        msg-type        [1] INTEGER (20),
        safe-body       [2] KRB-SAFE-BODY,
        cksum           [3] Checksum
}

KRB-SAFE-BODY   ::= SEQUENCE {
        user-data       [0] OCTET STRING,
        timestamp       [1] KerberosTime OPTIONAL,
        usec            [2] Microseconds OPTIONAL,
        seq-number      [3] UInt32 OPTIONAL,
        s-address       [4] HostAddress,
        r-address       [5] HostAddress OPTIONAL
}
```

pvno and msg-type
These fields are described above in Section 5.4.1. msg-type is KRB_SAFE.

safe-body
This field is a placeholder for the body of the KRB-SAFE message.

cksum
This field contains the checksum of the application data, computed with a key usage value of 15.

The checksum is computed over the encoding of the KRB-SAFE sequence. First, the cksum is set to a type zero, zero-length value, and the checksum is computed over the encoding of the KRB-SAFE sequence. Then the checksum is set to the result of that computation. Finally, the KRB-SAFE sequence is encoded again. This method, although different than the one specified in RFC 1510, corresponds to existing practice.

user-data
This field is part of the KRB_SAFE and KRB_PRIV messages, and contains the application-specific data that is being passed from the sender to the recipient.

timestamp
This field is part of the KRB_SAFE and KRB_PRIV messages. Its contents are the current time as known by the sender of the message. By checking the timestamp, the recipient of the message is able to make sure that it was recently generated, and is not a replay.

usec
This field is part of the KRB_SAFE and KRB_PRIV headers. It contains the microsecond part of the timestamp.

seq-number
This field is described above in Section 5.3.2.

s-address
Sender's address.

This field specifies the address in use by the sender of the message.

r-address
This field specifies the address in use by the recipient of the message. It MAY be omitted for some uses (such as broadcast protocols), but the recipient MAY arbitrarily reject such messages. This field, along with s-address, can be used to help detect messages that have been incorrectly or maliciously delivered to the wrong recipient.

5.7. KRB_PRIV Message Specification

This section specifies the format of a message that can be used by either side (client or server) of an application to send a message to its peer securely and privately. It presumes that a session key has previously been exchanged (for example, by using the KRB_AP_REQ/KRB_AP_REP messages).

5.7.1. KRB_PRIV Definition

The KRB_PRIV message contains user data encrypted in the Session Key. The message fields are as follows:

```
KRB-PRIV        ::= [APPLICATION 21] SEQUENCE {
        pvno            [0] INTEGER (5),
        msg-type        [1] INTEGER (21),
                        -- NOTE: there is no [2] tag
        enc-part        [3] EncryptedData -- EncKrbPrivPart
}

EncKrbPrivPart  ::= [APPLICATION 28] SEQUENCE {
        user-data       [0] OCTET STRING,
        timestamp       [1] KerberosTime OPTIONAL,
        usec            [2] Microseconds OPTIONAL,
        seq-number      [3] UInt32 OPTIONAL,
        s-address       [4] HostAddress -- sender's addr --,
        r-address       [5] HostAddress OPTIONAL -- recip's addr
}
```

pvno and msg-type
: These fields are described above in Section 5.4.1. msg-type is KRB_PRIV.

enc-part
: This field holds an encoding of the EncKrbPrivPart sequence encrypted under the session key, with a key usage value of 13. This encrypted encoding is used for the enc-part field of the KRB-PRIV message.

user-data, timestamp, usec, s-address, and r-address
: These fields are described above in Section 5.6.1.

seq-number
: This field is described above in Section 5.3.2.

5.8. KRB_CRED Message Specification

This section specifies the format of a message that can be used to send Kerberos credentials from one principal to another. It is presented here to encourage a common mechanism to be used by applications when forwarding tickets or providing proxies to subordinate servers. It presumes that a session key has already been exchanged, perhaps by using the KRB_AP_REQ/KRB_AP_REP messages.

5.8.1. KRB_CRED Definition

The KRB_CRED message contains a sequence of tickets to be sent and information needed to use the tickets, including the session key from each. The information needed to use the tickets is encrypted under an encryption key previously exchanged or transferred alongside the KRB_CRED message. The message fields are as follows:

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

EncKrbCredPart  ::= [APPLICATION 29] SEQUENCE {
        ticket-info     [0] SEQUENCE OF KrbCredInfo,
        nonce           [1] UInt32 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
}
```

pvno and msg-type
: These fields are described above in Section 5.4.1. msg-type is KRB_CRED.

tickets
: These are the tickets obtained from the KDC specifically for use by the intended recipient. Successive tickets are paired with the corresponding KrbCredInfo sequence from the enc-part of the KRB-CRED message.

enc-part
: This field holds an encoding of the EncKrbCredPart sequence encrypted under the session key shared by the sender and the intended recipient, with a key usage value of 14. This encrypted encoding is used for the enc-part field of the KRB-CRED message.

  Implementation note: Implementations of certain applications, most notably certain implementations of the Kerberos GSS-API mechanism, do not separately encrypt the contents of the EncKrbCredPart of the KRB-CRED message when sending it. In the case of those GSS-API mechanisms, this is not a security vulnerability, as the entire KRB-CRED message is itself embedded in an encrypted message.

nonce
: If practical, an application MAY require the inclusion of a nonce generated by the recipient of the message. If the same value is included as the nonce in the message, it provides evidence that the message is fresh and has not been replayed by an attacker. A nonce MUST NEVER be reused.

timestamp and usec
: These fields specify the time that the KRB-CRED message was generated. The time is used to provide assurance that the message is fresh.

s-address and r-address
: These fields are described above in Section 5.6.1. They are used optionally to provide additional assurance of the integrity of the KRB-CRED message.

key
: This field exists in the corresponding ticket passed by the KRB-CRED message and is used to pass the session key from the sender to the intended recipient. The field's encoding is described in Section 5.2.9.

The following fields are optional. If present, they can be associated with the credentials in the remote ticket file. If left out, then it is assumed that the recipient of the credentials already knows their values.

prealm and pname
The name and realm of the delegated principal identity.

flags, authtime, starttime, endtime, renew-till, srealm, sname, and caddr
These fields contain the values of the corresponding fields from the ticket found in the ticket field. Descriptions of the fields are identical to the descriptions in the KDC-REP message.

5.9. Error Message Specification

This section specifies the format for the KRB_ERROR message. The fields included in the message are intended to return as much information as possible about an error. It is not expected that all the information required by the fields will be available for all types of errors. If the appropriate information is not available when the message is composed, the corresponding field will be left out of the message.

Note that because the KRB_ERROR message is not integrity protected, it is quite possible for an intruder to synthesize or modify it. In particular, this means that the client SHOULD NOT use any fields in this message for security-critical purposes, such as setting a system clock or generating a fresh authenticator. The message can be useful, however, for advising a user on the reason for some failure.

5.9.1. KRB_ERROR Definition

The KRB_ERROR message consists of the following fields:

```
KRB-ERROR       ::= [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] Int32,
        crealm          [7] Realm OPTIONAL,
        cname           [8] PrincipalName OPTIONAL,
        realm           [9] Realm -- service realm --,
        sname           [10] PrincipalName -- service name --,
        e-text          [11] KerberosString OPTIONAL,
```

```
        e-data          [12] OCTET STRING OPTIONAL
}
```

pvno and msg-type
: These fields are described above in Section 5.4.1. msg-type is KRB_ERROR.

ctime and cusec
: These fields are described above in Section 5.5.2. If the values for these fields are known to the entity generating the error (as they would be if the KRB-ERROR is generated in reply to, e.g., a failed authentication service request), they should be populated in the KRB-ERROR. If the values are not available, these fields can be omitted.

stime
: This field contains the current time on the server. It is of type KerberosTime.

susec
: This field contains the microsecond part of the server's timestamp. Its value ranges from 0 to 999999. It appears along with stime. The two fields are used in conjunction to specify a reasonably accurate timestamp.

error-code
: This field contains the error code returned by Kerberos or the server when a request fails. To interpret the value of this field see the list of error codes in Section 7.5.9. Implementations are encouraged to provide for national language support in the display of error messages.

crealm, and cname
: These fields are described above in Section 5.3. When the entity generating the error knows these values, they should be populated in the KRB-ERROR. If the values are not known, the crealm and cname fields SHOULD be omitted.

realm and sname
: These fields are described above in Section 5.3.

e-text
: This field contains additional text to help explain the error code associated with the failed request (for example, it might include a principal name which was unknown).

e-data
This field contains additional data about the error for use by the application to help it recover from or handle the error. If the errorcode 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 otherwise. Whether defined by the implementation or in a future document, the e-data field MAY take the form of TYPED-DATA:

```
TYPED-DATA      ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
        data-type       [0] Int32,
        data-value      [1] OCTET STRING OPTIONAL
}
```

5.10. Application Tag Numbers

The following table lists the application class tag numbers used by various data types defined in this section.

Tag Number(s)	Type Name	Comments
0		unused
1	Ticket	PDU
2	Authenticator	non-PDU
3	EncTicketPart	non-PDU
4-9		unused
10	AS-REQ	PDU
11	AS-REP	PDU
12	TGS-REQ	PDU
13	TGS-REP	PDU

14	AP-REQ	PDU
15	AP-REP	PDU
16	RESERVED16	TGT-REQ (for user-to-user)
17	RESERVED17	TGT-REP (for user-to-user)
18-19		unused
20	KRB-SAFE	PDU
21	KRB-PRIV	PDU
22	KRB-CRED	PDU
23-24		unused
25	EncASRepPart	non-PDU
26	EncTGSRepPart	non-PDU
27	EncApRepPart	non-PDU
28	EncKrbPrivPart	non-PDU
29	EncKrbCredPart	non-PDU
30	KRB-ERROR	PDU

The ASN.1 types marked above as "PDU" (Protocol Data Unit) are the only ASN.1 types intended as top-level types of the Kerberos protocol, and are the only types that may be used as elements in another protocol that makes use of Kerberos.

6. Naming Constraints

6.1. Realm Names

Although realm names are encoded as GeneralStrings and technically a realm can select any name it chooses, interoperability across realm boundaries requires agreement on how realm names are to be assigned, and what information they imply.

To enforce these conventions, each realm MUST conform to the conventions itself, and it MUST require that any realms with which inter-realm keys are shared also conform to the conventions and require the same from its neighbors.

Kerberos realm names are case sensitive. Realm names that differ only in the case of the characters are not equivalent. There are presently three styles of realm names: domain, X500, and other. Examples of each style follow:

```
     domain:   ATHENA.MIT.EDU
       X500:   C=US/O=OSF
      other:   NAMETYPE:rest/of.name=without-restrictions
```

Domain style realm names MUST look like domain names: they consist of components separated by periods (.) and they contain neither colons (:) nor slashes (/). Though domain names themselves are case insensitive, in order for realms to match, the case must match as well. When establishing a new realm name based on an internet domain name it is recommended by convention that the characters be converted to uppercase.

X.500 names contain an equals sign (=) and cannot contain a colon (:) before the equals sign. The realm names for X.500 names will be string representations of the names with components separated by slashes. Leading and trailing slashes will not be included. Note that the slash separator is consistent with Kerberos implementations based on RFC 1510, but it is different from the separator recommended in RFC 2253.

Names that fall into the other category MUST begin with a prefix that contains no equals sign (=) or period (.), and the prefix MUST be followed by a colon (:) and the rest of the name. All prefixes expect those beginning with used. Presently none are assigned.

The reserved category includes strings that do not fall into the first three categories. All names in this category are reserved. It is unlikely that names will be assigned to this category unless there is a very strong argument for not using the 'other' category.

These rules guarantee that there will be no conflicts between the various name styles. The following additional constraints apply to the assignment of realm names in the domain and X.500 categories: either the name of a realm for the domain or X.500 formats must be used by the organization owning (to whom it was assigned) an Internet domain name or X.500 name, or, in the case that no such names are registered, authority to use a realm name MAY be derived from the authority of the parent realm. For example, if there is no domain name for E40.MIT.EDU, then the administrator of the MIT.EDU realm can authorize the creation of a realm with that name.

This is acceptable because the organization to which the parent is assigned is presumably the organization authorized to assign names to

its children in the X.500 and domain name systems as well. If the parent assigns a realm name without also registering it in the domain name or X.500 hierarchy, it is the parent's responsibility to make sure that in the future there will not exist a name identical to the realm name of the child unless it is assigned to the same entity as the realm name.

6.2. Principal Names

As was the case for realm names, conventions are needed to ensure that all agree on what information is implied by a principal name. The name-type field that is part of the principal name indicates the kind of information implied by the name. The name-type SHOULD be treated only as a hint to interpreting the meaning of a name. It is not significant when checking for equivalence. Principal names that differ only in the name-type identify the same principal. The name type does not partition the name space. 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). The following name types are defined:

Name Type	Value	Meaning
NT-UNKNOWN	0	Name type not known
NT-PRINCIPAL	1	Just the name of the principal as in DCE, or for users
NT-SRV-INST	2	Service and other unique instance (krbtgt)
NT-SRV-HST	3	Service with host name as instance (telnet, rcommands)
NT-SRV-XHST	4	Service with host as remaining components
NT-UID	5	Unique ID
NT-X500-PRINCIPAL	6	Encoded X.509 Distinguished name [RFC2253]
NT-SMTP-NAME	7	Name in form of SMTP email name (e.g., user@example.com)
NT-ENTERPRISE	10	Enterprise name - may be mapped to principal name

When a name implies no information other than its uniqueness at a particular time, the name type PRINCIPAL SHOULD be used. The principal name type SHOULD be used for users, and it might also be used for a unique server. If the name is a unique machine-generated ID that is guaranteed never to be reassigned, then the name type of UID SHOULD be used. (Note that it is generally a bad idea to reassign names of any type since stale entries might remain in access control lists.)

If the first component of a name identifies a service and the remaining components identify an instance of the service in a server-specified manner, then the name type of SRV-INST SHOULD be

used. An example of this name type is the Kerberos ticket-granting service whose name has a first component of krbtgt and a second component identifying the realm for which the ticket is valid.

If the first component of a name identifies a service and there is a single component following the service name identifying the instance as the host on which the server is running, then the name type SRV-HST SHOULD be used. This type is typically used for Internet services such as telnet and the Berkeley R commands. If the separate components of the host name appear as successive components following the name of the service, then the name type SRV-XHST SHOULD be used. This type might be used to identify servers on hosts with X.500 names, where the slash (/) might otherwise be ambiguous.

A name type of NT-X500-PRINCIPAL SHOULD be used when a name from an X.509 certificate is translated into a Kerberos name. The encoding of the X.509 name as a Kerberos principal shall conform to the encoding rules specified in RFC 2253.

A name type of SMTP allows a name to be of a form that resembles an SMTP email name. This name, including an "@" and a domain name, is used as the one component of the principal name.

A name type of UNKNOWN SHOULD be used when the form of the name is not known. When comparing names, a name of type UNKNOWN will match principals authenticated with names of any type. A principal authenticated with a name of type UNKNOWN, however, will only match other names of type UNKNOWN.

Names of any type with an initial component of 'krbtgt' are reserved for the Kerberos ticket-granting service. See Section 7.3 for the form of such names.

6.2.1. Name of Server Principals

The principal identifier for a server on a host will generally be composed of two parts: (1) the realm of the KDC with which the server is registered, and (2) a two-component name of type NT-SRV-HST, if the host name is an Internet domain name, or a multi-component name of type NT-SRV-XHST, if the name of the host is of a form (such as X.500) that allows slash (/) separators. The first component of the two- or multi-component name will identify the service, and the latter components will identify the host. Where the name of the host is not case sensitive (for example, with Internet domain names) the name of the host MUST be lowercase. If specified by the application protocol for services such as telnet and the Berkeley R commands that run with system privileges, the first component MAY be the string 'host' instead of a service-specific identifier.

7. Constants and Other Defined Values

7.1. Host Address Types

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.

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).

Internet (IPv6) Addresses

IPv6 addresses [RFC3513] 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.

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).

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).

Directional Addresses

Including the sender address in KRB_SAFE and KRB_PRIV messages is undesirable in many environments 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 KRB_AP_REQ message, then an address of 0 SHOULD be used. If the message is originated by the party to whom that KRB_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 KRB_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).

7.2. KDC Messaging: IP Transports

Kerberos defines two IP transport mechanisms for communication between clients and servers: UDP/IP and TCP/IP.

7.2.1. UDP/IP transport

Kerberos servers (KDCs) supporting IP transports MUST accept UDP requests and SHOULD listen for them 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 [7.2.3] 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 KRB_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 cannot 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.

7.2.2. TCP/IP Transport

Kerberos servers (KDCs) supporting IP transports MUST accept TCP requests and SHOULD listen for them 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 try a request initially using the UDP transport. Clients SHOULD use KDC discovery [7.2.3] 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 KRB_KDC_REQ message is sent to the KDC over a TCP stream, the response (KRB_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 follow-up. 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 any time 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 follow-up 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 (KRB_KDC_REQ) and response (KRB_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).

7.2.3. 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 that this method of storing configuration information presents problems with out-of-date information and scaling, especially when using cross-realm authentication. This section describes a method for using the Domain Name System [RFC1035] for storing KDC location information.

7.2.3.1. DNS vs. Kerberos: Case Sensitivity of Realm Names

In Kerberos, realm names are case sensitive. Although it is strongly encouraged that all realm names be all uppercase, this recommendation has not been adopted by all sites. Some sites use all lowercase names and other use mixed case. DNS, on the other hand, is case insensitive for queries. Because 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 lowercase characters be used in realm names.

7.2.3.2. Specifying KDC Location Information with DNS SRV records

KDC location information is to be stored using the DNS SRV RR [RFC2782]. 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 either "udp" or "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.

7.2.3.3. 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.
```

7.3. Name of the TGS

The principal identifier of the ticket-granting service shall be composed of three parts: the realm of the KDC issuing the TGS ticket, and a two-part name of type NT-SRV-INST, with the first part "krbtgt" and the second part the name of the realm that will accept the TGT. For example, a TGT issued by the ATHENA.MIT.EDU realm to be used to

get tickets from the ATHENA.MIT.EDU KDC has a principal identifier of "ATHENA.MIT.EDU" (realm), ("krbtgt", "ATHENA.MIT.EDU") (name). A TGT issued by the ATHENA.MIT.EDU realm to be used to get tickets from the MIT.EDU realm has a principal identifier of "ATHENA.MIT.EDU" (realm), ("krbtgt", "MIT.EDU") (name).

7.4. OID Arc for KerberosV5

This OID MAY be used to identify Kerberos protocol messages encapsulated in other protocols. It also designates the OID arc for KerberosV5-related OIDs assigned by future IETF action.
Implementation 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)
}
```

Assignment of OIDs beneath the id-krb5 arc must be obtained by contacting the registrar for the id-krb5 arc, or its designee. At the time of the issuance of this RFC, such registrations can be obtained by contacting krb5-oid-registrar@mit.edu.

7.5. Protocol Constants and Associated Values

The following tables list constants used in the protocol and define their meanings. In the "specification" section, ranges are specified that limit the values of constants for which values are defined here. This allows implementations to make assumptions about the maximum values that will be received for these constants. Implementations receiving values outside the range specified in the "specification" section MAY reject the request, but they MUST recover cleanly.

7.5.1. Key Usage Numbers

The encryption and checksum specifications in [RFC3961] require as input a "key usage number", to alter the encryption key used in any specific message in order to make certain types of cryptographic attack more difficult. These are the key usage values assigned in this document:

1. AS-REQ PA-ENC-TIMESTAMP padata timestamp, encrypted with the client key (Section 5.2.7.2)

2. AS-REP Ticket and TGS-REP Ticket (includes TGS session key or application session key), encrypted with the service key (Section 5.3)
3. AS-REP encrypted part (includes TGS session key or application session key), encrypted with the client key (Section 5.4.2)
4. TGS-REQ KDC-REQ-BODY AuthorizationData, encrypted with the TGS session key (Section 5.4.1)
5. TGS-REQ KDC-REQ-BODY AuthorizationData, encrypted with the TGS authenticator subkey (Section 5.4.1)
6. TGS-REQ PA-TGS-REQ padata AP-REQ Authenticator cksum, keyed with the TGS session key (Section 5.5.1)
7. TGS-REQ PA-TGS-REQ padata AP-REQ Authenticator (includes TGS authenticator subkey), encrypted with the TGS session key (Section 5.5.1)
8. TGS-REP encrypted part (includes application session key), encrypted with the TGS session key (Section 5.4.2)
9. TGS-REP encrypted part (includes application session key), encrypted with the TGS authenticator subkey (Section 5.4.2)
10. AP-REQ Authenticator cksum, keyed with the application session key (Section 5.5.1)
11. AP-REQ Authenticator (includes application authenticator subkey), encrypted with the application session key (Section 5.5.1)
12. AP-REP encrypted part (includes application session subkey), encrypted with the application session key (Section 5.5.2)
13. KRB-PRIV encrypted part, encrypted with a key chosen by the application (Section 5.7.1)
14. KRB-CRED encrypted part, encrypted with a key chosen by the application (Section 5.8.1)
15. KRB-SAFE cksum, keyed with a key chosen by the application (Section 5.6.1)

16-18. Reserved for future use in Kerberos and related protocols.

19. AD-KDC-ISSUED checksum (ad-checksum in 5.2.6.4)

20-21. Reserved for future use in Kerberos and related protocols.

22-25. Reserved for use in the Kerberos Version 5 GSS-API mechanisms [RFC4121].

26-511. Reserved for future use in Kerberos and related protocols.

512-1023. Reserved for uses internal to a Kerberos implementation.

1024. Encryption for application use in protocols that do not specify key usage values

1025. Checksums for application use in protocols that do not specify key usage values
1026-2047. Reserved for application use.

7.5.2. PreAuthentication Data Types

Padata and Data Type	Padata-type Value	Comment
PA-TGS-REQ	1	
PA-ENC-TIMESTAMP	2	
PA-PW-SALT	3	
[reserved]	4	
PA-ENC-UNIX-TIME	5	(deprecated)
PA-SANDIA-SECUREID	6	
PA-SESAME	7	
PA-OSF-DCE	8	
PA-CYBERSAFE-SECUREID	9	
PA-AFS3-SALT	10	
PA-ETYPE-INFO	11	
PA-SAM-CHALLENGE	12	(sam/otp)
PA-SAM-RESPONSE	13	(sam/otp)
PA-PK-AS-REQ_OLD	14	(pkinit)
PA-PK-AS-REP_OLD	15	(pkinit)
PA-PK-AS-REQ	16	(pkinit)
PA-PK-AS-REP	17	(pkinit)
PA-ETYPE-INFO2	19	(replaces pa-etype-info)
PA-USE-SPECIFIED-KVNO	20	
PA-SAM-REDIRECT	21	(sam/otp)
PA-GET-FROM-TYPED-DATA	22	(embedded in typed data)
TD-PADATA	22	(embeds padata)
PA-SAM-ETYPE-INFO	23	(sam/otp)
PA-ALT-PRINC	24	(crawdad@fnal.gov)
PA-SAM-CHALLENGE2	30	(kenh@pobox.com)
PA-SAM-RESPONSE2	31	(kenh@pobox.com)
PA-EXTRA-TGT	41	Reserved extra TGT
TD-PKINIT-CMS-CERTIFICATES	101	CertificateSet from CMS
TD-KRB-PRINCIPAL	102	PrincipalName
TD-KRB-REALM	103	Realm
TD-TRUSTED-CERTIFIERS	104	from PKINIT
TD-CERTIFICATE-INDEX	105	from PKINIT
TD-APP-DEFINED-ERROR	106	application specific
TD-REQ-NONCE	107	INTEGER
TD-REQ-SEQ	108	INTEGER
PA-PAC-REQUEST	128	(jbrezak@exchange.microsoft.com)

7.5.3. Address Types

Address Type	Value
IPv4	2
Directional	3
ChaosNet	5
XNS	6
ISO	7
DECNET Phase IV	12
AppleTalk DDP	16
NetBios	20
IPv6	24

7.5.4. Authorization Data Types

Authorization Data Type	Ad-type Value	
AD-IF-RELEVANT	1	
AD-INTENDED-FOR-SERVER	2	
AD-INTENDED-FOR-APPLICATION-CLASS	3	
AD-KDC-ISSUED	4	
AD-AND-OR	5	
AD-MANDATORY-TICKET-EXTENSIONS	6	
AD-IN-TICKET-EXTENSIONS	7	
AD-MANDATORY-FOR-KDC	8	
Reserved values	9-63	
OSF-DCE	64	
SESAME	65	
AD-OSF-DCE-PKI-CERTID	66	(hemsath@us.ibm.com)
AD-WIN2K-PAC	128	(jbrezak@exchange.microsoft.com)
AD-ETYPE-NEGOTIATION	129	(lzhu@windows.microsoft.com)

7.5.5. Transited Encoding Types

Transited Encoding Type	Tr-type Value
DOMAIN-X500-COMPRESS	1
Reserved values	All others

7.5.6. Protocol Version Number

Label	Value	Meaning or MIT Code
pvno	5	Current Kerberos protocol version number

7.5.7. Kerberos Message Types

Message Type	Value	Meaning
KRB_AS_REQ	10	Request for initial authentication
KRB_AS_REP	11	Response to KRB_AS_REQ request
KRB_TGS_REQ	12	Request for authentication based on TGT
KRB_TGS_REP	13	Response to KRB_TGS_REQ request
KRB_AP_REQ	14	Application request to server
KRB_AP_REP	15	Response to KRB_AP_REQ_MUTUAL
KRB_RESERVED16	16	Reserved for user-to-user krb_tgt_request
KRB_RESERVED17	17	Reserved for user-to-user krb_tgt_reply
KRB_SAFE	20	Safe (checksummed) application message
KRB_PRIV	21	Private (encrypted) application message
KRB_CRED	22	Private (encrypted) message to forward credentials
KRB_ERROR	30	Error response

7.5.8. Name Types

Name Type	Value	Meaning
KRB_NT_UNKNOWN	0	Name type not known
KRB_NT_PRINCIPAL	1	Just the name of the principal as in DCE, or for users
KRB_NT_SRV_INST	2	Service and other unique instance (krbtgt)
KRB_NT_SRV_HST	3	Service with host name as instance (telnet, rcommands)
KRB_NT_SRV_XHST	4	Service with host as remaining components
KRB_NT_UID	5	Unique ID
KRB_NT_X500_PRINCIPAL	6	Encoded X.509 Distinguished name [RFC2253]
KRB_NT_SMTP_NAME	7	Name in form of SMTP email name (e.g., user@example.com)
KRB_NT_ENTERPRISE	10	Enterprise name; may be mapped to principal name

7.5.9. Error Codes

Error Code	Value	Meaning
KDC_ERR_NONE	0	No error
KDC_ERR_NAME_EXP	1	Client’s entry in database has expired
KDC_ERR_SERVICE_EXP	2	Server’s entry in database has expired
KDC_ERR_BAD_PVNO	3	Requested protocol version number not supported

KDC_ERR_C_OLD_MAST_KVNO	4	Client's key encrypted in old master key
KDC_ERR_S_OLD_MAST_KVNO	5	Server's key encrypted in old master key
KDC_ERR_C_PRINCIPAL_UNKNOWN	6	Client not found in Kerberos database
KDC_ERR_S_PRINCIPAL_UNKNOWN	7	Server not found in Kerberos database
KDC_ERR_PRINCIPAL_NOT_UNIQUE	8	Multiple principal entries in database
KDC_ERR_NULL_KEY	9	The client or server has a null key
KDC_ERR_CANNOT_POSTDATE	10	Ticket not eligible for postdating
KDC_ERR_NEVER_VALID	11	Requested starttime is later than end time
KDC_ERR_POLICY	12	KDC policy rejects request
KDC_ERR_BADOPTION	13	KDC cannot accommodate requested option
KDC_ERR_ETYPE_NOSUPP	14	KDC has no support for encryption type
KDC_ERR_SUMTYPE_NOSUPP	15	KDC has no support for checksum type
KDC_ERR_PADATA_TYPE_NOSUPP	16	KDC has no support for padata type
KDC_ERR_TRTYPE_NOSUPP	17	KDC has no support for transited type
KDC_ERR_CLIENT_REVOKED	18	Clients credentials have been revoked
KDC_ERR_SERVICE_REVOKED	19	Credentials for server have been revoked
KDC_ERR_TGT_REVOKED	20	TGT has been revoked
KDC_ERR_CLIENT_NOTYET	21	Client not yet valid; try again later
KDC_ERR_SERVICE_NOTYET	22	Server not yet valid; try again later
KDC_ERR_KEY_EXPIRED	23	Password has expired; change password to reset
KDC_ERR_PREAUTH_FAILED	24	Pre-authentication information was invalid
KDC_ERR_PREAUTH_REQUIRED	25	Additional pre-authentication required
KDC_ERR_SERVER_NOMATCH	26	Requested server and ticket don't match
KDC_ERR_MUST_USE_USER2USER	27	Server principal valid for user2user only
KDC_ERR_PATH_NOT_ACCEPTED	28	KDC Policy rejects transited path

KDC_ERR_SVC_UNAVAILABLE	29	A service is not available
KRB_AP_ERR_BAD_INTEGRITY	31	Integrity check on decrypted field failed
KRB_AP_ERR_TKT_EXPIRED	32	Ticket expired
KRB_AP_ERR_TKT_NYV	33	Ticket not yet valid
KRB_AP_ERR_REPEAT	34	Request is a replay
KRB_AP_ERR_NOT_US	35	The ticket isn't for us
KRB_AP_ERR_BADMATCH	36	Ticket and authenticator don't match
KRB_AP_ERR_SKEW	37	Clock skew too great
KRB_AP_ERR_BADADDR	38	Incorrect net address
KRB_AP_ERR_BADVERSION	39	Protocol version mismatch
KRB_AP_ERR_MSG_TYPE	40	Invalid msg type
KRB_AP_ERR_MODIFIED	41	Message stream modified
KRB_AP_ERR_BADORDER	42	Message out of order
KRB_AP_ERR_BADKEYVER	44	Specified version of key is not available
KRB_AP_ERR_NOKEY	45	Service key not available
KRB_AP_ERR_MUT_FAIL	46	Mutual authentication failed
KRB_AP_ERR_BADDIRECTION	47	Incorrect message direction
KRB_AP_ERR_METHOD	48	Alternative authentication method required
KRB_AP_ERR_BADSEQ	49	Incorrect sequence number in message
KRB_AP_ERR_INAPP_CKSUM	50	Inappropriate type of checksum in message
KRB_AP_PATH_NOT_ACCEPTED	51	Policy rejects transited path
KRB_ERR_RESPONSE_TOO_BIG	52	Response too big for UDP; retry with TCP
KRB_ERR_GENERIC	60	Generic error (description in e-text)
KRB_ERR_FIELD_TOOLONG	61	Field is too long for this implementation
KDC_ERROR_CLIENT_NOT_TRUSTED	62	Reserved for PKINIT
KDC_ERROR_KDC_NOT_TRUSTED	63	Reserved for PKINIT
KDC_ERROR_INVALID_SIG	64	Reserved for PKINIT
KDC_ERR_KEY_TOO_WEAK	65	Reserved for PKINIT
KDC_ERR_CERTIFICATE_MISMATCH	66	Reserved for PKINIT
KRB_AP_ERR_NO_TGT	67	No TGT available to validate USER-TO-USER
KDC_ERR_WRONG_REALM	68	Reserved for future use
KRB_AP_ERR_USER_TO_USER_REQUIRED	69	Ticket must be for USER-TO-USER
KDC_ERR_CANT_VERIFY_CERTIFICATE	70	Reserved for PKINIT
KDC_ERR_INVALID_CERTIFICATE	71	Reserved for PKINIT
KDC_ERR_REVOKED_CERTIFICATE	72	Reserved for PKINIT

KDC_ERR_REVOCATION_STATUS_UNKNOWN	73	Reserved for PKINIT
KDC_ERR_REVOCATION_STATUS_UNAVAILABLE	74	Reserved for PKINIT
KDC_ERR_CLIENT_NAME_MISMATCH	75	Reserved for PKINIT
KDC_ERR_KDC_NAME_MISMATCH	76	Reserved for PKINIT

8. Interoperability Requirements

Version 5 of the Kerberos protocol supports a myriad of options. Among these are multiple encryption and checksum types; alternative encoding schemes for the transited field; optional mechanisms for pre-authentication; the handling of tickets with no addresses; options for mutual authentication; user-to-user authentication; support for proxies; the format of realm names; the handling of authorization data; and forwarding, postdating, and renewing tickets.

In order to ensure the interoperability of realms, it is necessary to define a minimal configuration that must be supported by all implementations. This minimal configuration is subject to change as technology does. For example, if at some later date it is discovered that one of the required encryption or checksum algorithms is not secure, it will be replaced.

8.1. Specification 2

This section defines the second specification of these options. Implementations which are configured in this way can be said to support Kerberos Version 5 Specification 2 (5.2). Specification 1 (deprecated) may be found in RFC 1510.

Transport

TCP/IP and UDP/IP transport MUST be supported by clients and KDCs claiming conformance to specification 2.

Encryption and Checksum Methods

The following encryption and checksum mechanisms MUST be supported:

Encryption: AES256-CTS-HMAC-SHA1-96 [RFC3962]
Checksums: HMAC-SHA1-96-AES256 [RFC3962]

Implementations SHOULD support other mechanisms as well, but the additional mechanisms may only be used when communicating with principals known to also support them. The following mechanisms from [RFC3961] and [RFC3962] SHOULD be supported:

Encryption: AES128-CTS-HMAC-SHA1-96, DES-CBC-MD5, DES3-CBC-SHA1-KD
Checksums: DES-MD5, HMAC-SHA1-DES3-KD, HMAC-SHA1-96-AES128

Implementations MAY support other mechanisms as well, but the additional mechanisms may only be used when communicating with principals known to support them also.

Implementation note: Earlier implementations of Kerberos generate messages using the CRC-32 and RSA-MD5 checksum methods. For interoperability with these earlier releases, implementors MAY consider supporting these checksum methods but should carefully analyze the security implications to limit the situations within which these methods are accepted.

Realm Names

All implementations MUST understand hierarchical realms in both the Internet Domain and the X.500 style. When a TGT for an unknown realm is requested, the KDC MUST be able to determine the names of the intermediate realms between the KDCs realm and the requested realm.

Transited Field Encoding

DOMAIN-X500-COMPRESS (described in Section 3.3.3.2) MUST be supported. Alternative encodings MAY be supported, but they may only be used when that encoding is supported by ALL intermediate realms.

Pre-authentication Methods

The TGS-REQ method MUST be supported. It is not used on the initial request. The PA-ENC-TIMESTAMP method MUST be supported by clients, but whether it is enabled by default MAY be determined on a realm-by-realm basis. If the method is not used in the initial request and the error KDC_ERR_PREAUTH_REQUIRED is returned specifying PA-ENC-TIMESTAMP as an acceptable method, the client SHOULD retry the initial request using the PA-ENC-TIMESTAMP pre-authentication method. Servers need not support the PA-ENC-TIMESTAMP method, but if it is not supported the server SHOULD ignore the presence of PA-ENC-TIMESTAMP pre-authentication in a request.

The ETYPE-INFO2 method MUST be supported; this method is used to communicate the set of supported encryption types, and corresponding salt and string to key parameters. The ETYPE-INFO method SHOULD be supported for interoperability with older implementation.

Mutual Authentication

Mutual authentication (via the KRB_AP_REP message) MUST be supported.

Ticket Addresses and Flags

All KDCs MUST pass through tickets that carry no addresses (i.e., if a TGT contains no addresses, the KDC will return derivative tickets). Implementations SHOULD default to requesting addressless tickets, as this significantly increases interoperability with network address translation. In some cases, realms or application servers MAY require that tickets have an address.

Implementations SHOULD accept directional address type for the KRB_SAFE and KRB_PRIV message and SHOULD include directional addresses in these messages when other address types are not available.

Proxies and forwarded tickets MUST be supported. Individual realms and application servers can set their own policy on when such tickets will be accepted.

All implementations MUST recognize renewable and postdated tickets, but they need not actually implement them. If these options are not supported, the starttime and endtime in the ticket SHALL specify a ticket's entire useful life. When a postdated ticket is decoded by a server, all implementations SHALL make the presence of the postdated flag visible to the calling server.

User-to-User Authentication

Support for user-to-user authentication (via the ENC-TKT-IN-SKEY KDC option) MUST be provided by implementations, but individual realms MAY decide as a matter of policy to reject such requests on a per-principal or realm-wide basis.

Authorization Data

Implementations MUST pass all authorization data subfields from TGTs to any derivative tickets unless they are directed to suppress a subfield as part of the definition of that registered subfield type. (It is never incorrect to pass on a subfield, and no registered subfield types presently specify suppression at the KDC.)

Implementations MUST make the contents of any authorization data subfields available to the server when a ticket is used. Implementations are not required to allow clients to specify the contents of the authorization data fields.

Constant Ranges

All protocol constants are constrained to 32-bit (signed) values unless further constrained by the protocol definition. This limit is provided to allow implementations to make assumptions about the maximum values that will be received for these constants. Implementations receiving values outside this range MAY reject the request, but they MUST recover cleanly.

8.2. Recommended KDC Values

Following is a list of recommended values for a KDC configuration.

Minimum lifetime	5 minutes
Maximum renewable lifetime	1 week
Maximum ticket lifetime	1 day
Acceptable clock skew	5 minutes
Empty addresses	Allowed
Proxiable, etc.	Allowed

9. IANA Considerations

Section 7 of this document specifies protocol constants and other defined values required for the interoperability of multiple implementations. Until a subsequent RFC specifies otherwise, or the Kerberos working group is shut down, allocations of additional protocol constants and other defined values required for extensions to the Kerberos protocol will be administered by the Kerberos working group. Following the recommendations outlined in [RFC2434], guidance is provided to the IANA as follows:

"reserved" realm name types in Section 6.1 and "other" realm types except those beginning with "X-" or "x-" will not be registered without IETF standards action, at which point guidelines for further assignment will be specified. Realm name types beginning with "X-" or "x-" are for private use.

For host address types described in Section 7.1, negative values are for private use. Assignment of additional positive numbers is subject to review by the Kerberos working group or other expert review.

Additional key usage numbers, as defined in Section 7.5.1, will be assigned subject to review by the Kerberos working group or other expert review.

Additional preauthentication data type values, as defined in section 7.5.2, will be assigned subject to review by the Kerberos working group or other expert review.

Additional authorization data types as defined in Section 7.5.4, will be assigned subject to review by the Kerberos working group or other expert review. Although it is anticipated that there may be significant demand for private use types, provision is intentionally not made for a private use portion of the namespace because conflicts between privately assigned values could have detrimental security implications.

Additional transited encoding types, as defined in Section 7.5.5, present special concerns for interoperability with existing implementations. As such, such assignments will only be made by standards action, except that the Kerberos working group or another other working group with competent jurisdiction may make preliminary assignments for documents that are moving through the standards process.

Additional Kerberos message types, as described in Section 7.5.7, will be assigned subject to review by the Kerberos working group or other expert review.

Additional name types, as described in Section 7.5.8, will be assigned subject to review by the Kerberos working group or other expert review.

Additional error codes described in Section 7.5.9 will be assigned subject to review by the Kerberos working group or other expert review.

10. Security Considerations

As an authentication service, Kerberos provides a means of verifying the identity of principals on a network. By itself, Kerberos does not provide authorization. Applications should not accept the issuance of a service ticket by the Kerberos server as granting authority to use the service, since such applications may become vulnerable to the bypass of this authorization check in an environment where they inter-operate with other KDCs or where other options for application authentication are provided.

Denial of service attacks are not solved with Kerberos. There are places in the protocols where an intruder can prevent an application from participating in the proper authentication steps. Because authentication is a required step for the use of many services, successful denial of service attacks on a Kerberos server might result in the denial of other network services that rely on Kerberos for authentication. Kerberos is vulnerable to many kinds of denial of service attacks: those on the network, which would prevent clients from contacting the KDC; those on the domain name system, which could prevent a client from finding the IP address of the Kerberos server; and those by overloading the Kerberos KDC itself with repeated requests.

Interoperability conflicts caused by incompatible character-set usage (see 5.2.1) can result in denial of service for clients that utilize character-sets in Kerberos strings other than those stored in the KDC database.

Authentication servers maintain a database of principals (i.e., users and servers) and their secret keys. The security of the authentication server machines is critical. The breach of security of an authentication server will compromise the security of all servers that rely upon the compromised KDC, and will compromise the authentication of any principals registered in the realm of the compromised KDC.

Principals must keep their secret keys secret. If an intruder somehow steals a principal's key, it will be able to masquerade as that principal or impersonate any server to the legitimate principal.

Password-guessing attacks are not solved by Kerberos. If a user chooses a poor password, it is possible for an attacker to successfully mount an off-line dictionary attack by repeatedly attempting to decrypt, with successive entries from a dictionary, messages obtained that are encrypted under a key derived from the user's password.

Unless pre-authentication options are required by the policy of a realm, the KDC will not know whether a request for authentication succeeds. An attacker can request a reply with credentials for any principal. These credentials will likely not be of much use to the attacker unless it knows the client's secret key, but the availability of the response encrypted in the client's secret key provides the attacker with ciphertext that may be used to mount brute force or dictionary attacks to decrypt the credentials, by guessing the user's password. For this reason it is strongly encouraged that Kerberos realms require the use of pre-authentication. Even with

pre-authentication, attackers may try brute force or dictionary attacks against credentials that are observed by eavesdropping on the network.

Because a client can request a ticket for any server principal and can attempt a brute force or dictionary attack against the server principal's key using that ticket, it is strongly encouraged that keys be randomly generated (rather than generated from passwords) for any principals that are usable as the target principal for a KRB_TGS_REQ or KRB_AS_REQ messages. [RFC4086]

Although the DES-CBC-MD5 encryption method and DES-MD5 checksum methods are listed as SHOULD be implemented for backward compatibility, the single DES encryption algorithm on which these are based is weak, and stronger algorithms should be used whenever possible.

Each host on the network must have a clock that is loosely synchronized to the time of the other hosts; this synchronization is used to reduce the bookkeeping needs of application servers when they do replay detection. The degree of "looseness" can be configured on a per-server basis, but it is typically on the order of 5 minutes. If the clocks are synchronized over the network, the clock synchronization protocol MUST itself be secured from network attackers.

Principal identifiers must not recycled on a short-term basis. A typical mode of access control will use access control lists (ACLs) to grant permissions to particular principals. If a stale ACL entry remains for a deleted principal and the principal identifier is reused, the new principal will inherit rights specified in the stale ACL entry. By not reusing principal identifiers, the danger of inadvertent access is removed.

Proper decryption of an KRB_AS_REP message from the KDC is not sufficient for the host to verify the identity of the user; the user and an attacker could cooperate to generate a KRB_AS_REP format message that decrypts properly but is not from the proper KDC. To authenticate a user logging on to a local system, the credentials obtained in the AS exchange may first be used in a TGS exchange to obtain credentials for a local server. Those credentials must then be verified by a local server through successful completion of the Client/Server exchange.

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

Kerberos credentials contain clear-text information identifying the principals to which they apply. If privacy of this information is needed, this exchange should itself be encapsulated in a protocol providing for confidentiality on the exchange of these credentials.

Applications must take care to protect communications subsequent to authentication, either by using the KRB_PRIV or KRB_SAFE messages as appropriate, or by applying their own confidentiality or integrity mechanisms on such communications. Completion of the KRB_AP_REQ and KRB_AP_REP exchange without subsequent use of confidentiality and integrity mechanisms provides only for authentication of the parties to the communication and not confidentiality and integrity of the subsequent communication. Applications applying confidentiality and integrity protection mechanisms other than KRB_PRIV and KRB_SAFE must make sure that the authentication step is appropriately linked with the protected communication channel that is established by the application.

Unless the application server provides its own suitable means to protect against replay (for example, a challenge-response sequence initiated by the server after authentication, or use of a server-generated encryption subkey), the server must utilize a replay cache to remember any authenticator presented within the allowable clock skew. All services sharing a key need to use the same replay cache. If separate replay caches are used, then an authenticator used with one such service could later be replayed to a different service with the same service principal.

If a server loses track of authenticators presented within the allowable clock skew, it must reject all requests until the clock skew interval has passed, providing assurance that any lost or replayed authenticators will fall outside the allowable clock skew and can no longer be successfully replayed.

Implementations of Kerberos should not use untrusted directory servers to determine the realm of a host. To allow this would allow the compromise of the directory server to enable an attacker to direct the client to accept authentication with the wrong principal (i.e., one with a similar name, but in a realm with which the legitimate host was not registered).

Implementations of Kerberos must not use DNS to map one name to another (canonicalize) in order to determine the host part of the principal name with which one is to communicate. To allow this canonicalization would allow a compromise of the DNS to result in a client obtaining credentials and correctly authenticating to the

wrong principal. Though the client will know who it is communicating with, it will not be the principal with which it intended to communicate.

If the Kerberos server returns a TGT for a realm 'closer' than the desired realm, the client may use local policy configuration to verify that the authentication path used is an acceptable one. Alternatively, a client may choose its own authentication path rather than rely on the Kerberos server to select one. In either case, any policy or configuration information used to choose or validate authentication paths, whether by the Kerberos server or client, must be obtained from a trusted source.

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 the user's, application server's, or KDC's key is subsequently discovered. This is because the session key used to encrypt such messages, when transmitted over the network, is encrypted in the key of the application server. It is also encrypted under the session key from the user's TGT when it is returned to the user in the KRB_TGS_REP message. The session key from the TGT is sent to the user in the KRB_AS_REP message encrypted in the user's secret key and embedded in the TGT, which was encrypted in the key of the KDC. Applications 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.

11. Acknowledgements

This document is a revision to RFC 1510 which was co-authored with John Kohl. The specification of the Kerberos protocol described in this document is the result of many years of effort. Over this period, many individuals have contributed to the definition of the protocol and to the writing of the specification. Unfortunately, it is not possible to list all contributors as authors of this document, though there are many not listed who are authors in spirit, including those who contributed text for parts of some sections, who contributed to the design of parts of the protocol, and who contributed significantly to the discussion of the protocol in the IETF common authentication technology (CAT) and Kerberos working groups.

Among those contributing to the development and specification of Kerberos were Jeffrey Altman, John Brezak, Marc Colan, Johan Danielsson, Don Davis, Doug Engert, Dan Geer, Paul Hill, John Kohl, Marc Horowitz, Matt Hur, Jeffrey Hutzelman, Paul Leach, John Linn, Ari Medvinsky, Sasha Medvinsky, Steve Miller, Jon Rochlis, Jerome Saltzer, Jeffrey Schiller, Jennifer Steiner, Ralph Swick, Mike Swift, Jonathan Trostle, Theodore Ts'o, Brian Tung, Jacques Vidrine, Assar Westerlund, and Nicolas Williams. Many other members of MIT Project Athena, the MIT networking group, and the Kerberos and CAT working groups of the IETF contributed but are not listed.

A. ASN.1 module

```
KerberosV5Spec2 {
        iso(1) identified-organization(3) dod(6) internet(1)
        security(5) kerberosV5(2) modules(4) krb5spec2(2)
} 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)
}

Int32           ::= INTEGER (-2147483648..2147483647)
                    -- signed values representable in 32 bits

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

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

KerberosString  ::= GeneralString (IA5String)

Realm           ::= KerberosString

PrincipalName   ::= SEQUENCE {
        name-type       [0] Int32,
        name-string     [1] SEQUENCE OF KerberosString
}

KerberosTime    ::= GeneralizedTime -- with no fractional seconds

HostAddress     ::= SEQUENCE  {
        addr-type       [0] Int32,
        address         [1] OCTET STRING
}

-- NOTE: HostAddresses is always used as an OPTIONAL field and
-- should not be empty.
HostAddresses   -- NOTE: subtly different from rfc1510,
```

```
                -- but has a value mapping and encodes the same
        ::= SEQUENCE OF HostAddress

-- NOTE: AuthorizationData is always used as an OPTIONAL field and
-- should not be empty.
AuthorizationData       ::= SEQUENCE OF SEQUENCE {
        ad-type         [0] Int32,
        ad-data         [1] OCTET STRING
}

PA-DATA         ::= SEQUENCE {
        -- NOTE: first tag is [1], not [0]
        padata-type     [1] Int32,
        padata-value    [2] OCTET STRING -- might be encoded AP-REQ
}

KerberosFlags   ::= BIT STRING (SIZE (32..MAX))
                    -- minimum number of bits shall be sent,
                    -- but no fewer than 32

EncryptedData   ::= SEQUENCE {
        etype   [0] Int32 -- EncryptionType --,
        kvno    [1] UInt32 OPTIONAL,
        cipher  [2] OCTET STRING -- ciphertext
}

EncryptionKey   ::= SEQUENCE {
        keytype         [0] Int32 -- actually encryption type --,
        keyvalue        [1] OCTET STRING
}

Checksum        ::= SEQUENCE {
        cksumtype       [0] Int32,
        checksum        [1] OCTET STRING
}

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

-- Encrypted part of ticket
EncTicketPart   ::= [APPLICATION 3] SEQUENCE {
        flags                   [0] TicketFlags,
        key                     [1] EncryptionKey,
        crealm                  [2] Realm,
```

```
        cname                   [3] PrincipalName,
        transited               [4] TransitedEncoding,
        authtime                [5] KerberosTime,
        starttime               [6] KerberosTime OPTIONAL,
        endtime                 [7] KerberosTime,
        renew-till              [8] KerberosTime OPTIONAL,
        caddr                   [9] HostAddresses OPTIONAL,
        authorization-data      [10] AuthorizationData OPTIONAL
}

-- encoded Transited field
TransitedEncoding       ::= SEQUENCE {
        tr-type         [0] Int32 -- must be registered --,
        contents        [1] OCTET STRING
}

TicketFlags     ::= KerberosFlags
        -- 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),
-- the following are new since 1510
        -- transited-policy-checked(12),
        -- ok-as-delegate(13)

AS-REQ          ::= [APPLICATION 10] KDC-REQ

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

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

KDC-REQ-BODY    ::= 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,
        rtime                   [6] KerberosTime OPTIONAL,
        nonce                   [7] UInt32,
        etype                   [8] SEQUENCE OF Int32 -- EncryptionType
                                    -- in preference order --,
        addresses               [9] HostAddresses OPTIONAL,
        enc-authorization-data  [10] EncryptedData OPTIONAL
                                    -- AuthorizationData --,
        additional-tickets      [11] SEQUENCE OF Ticket OPTIONAL
                                        -- NOTE: not empty
}

KDCOptions      ::= KerberosFlags
        -- reserved(0),
        -- forwardable(1),
        -- forwarded(2),
        -- proxiable(3),
        -- proxy(4),
        -- allow-postdate(5),
        -- postdated(6),
        -- unused7(7),
        -- renewable(8),
        -- unused9(9),
        -- unused10(10),
        -- opt-hardware-auth(11),
        -- unused12(12),
        -- unused13(13),
-- 15 is reserved for canonicalize
        -- unused15(15),
-- 26 was unused in 1510
        -- disable-transited-check(26),
--
        -- renewable-ok(27),
        -- enc-tkt-in-skey(28),
        -- renew(30),
        -- validate(31)

AS-REP          ::= [APPLICATION 11] KDC-REP

TGS-REP         ::= [APPLICATION 13] KDC-REP
```

```
KDC-REP         ::= SEQUENCE {
        pvno            [0] INTEGER (5),
        msg-type        [1] INTEGER (11 -- AS -- | 13 -- TGS --),
        padata          [2] SEQUENCE OF PA-DATA OPTIONAL
                                -- NOTE: not empty --,
        crealm          [3] Realm,
        cname           [4] PrincipalName,
        ticket          [5] Ticket,
        enc-part        [6] EncryptedData
                                -- EncASRepPart or EncTGSRepPart,
                                -- as appropriate
}

EncASRepPart    ::= [APPLICATION 25] EncKDCRepPart

EncTGSRepPart   ::= [APPLICATION 26] EncKDCRepPart

EncKDCRepPart   ::= SEQUENCE {
        key             [0] EncryptionKey,
        last-req        [1] LastReq,
        nonce           [2] UInt32,
        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
}

LastReq         ::=     SEQUENCE OF SEQUENCE {
        lr-type         [0] Int32,
        lr-value        [1] KerberosTime
}

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

APOptions       ::= KerberosFlags
        -- reserved(0),
        -- use-session-key(1),
```

```
        -- mutual-required(2)

-- Unencrypted authenticator
Authenticator   ::= [APPLICATION 2] SEQUENCE  {
        authenticator-vno       [0] INTEGER (5),
        crealm                  [1] Realm,
        cname                   [2] PrincipalName,
        cksum                   [3] Checksum OPTIONAL,
        cusec                   [4] Microseconds,
        ctime                   [5] KerberosTime,
        subkey                  [6] EncryptionKey OPTIONAL,
        seq-number              [7] UInt32 OPTIONAL,
        authorization-data      [8] AuthorizationData OPTIONAL
}

AP-REP          ::= [APPLICATION 15] SEQUENCE {
        pvno            [0] INTEGER (5),
        msg-type        [1] INTEGER (15),
        enc-part        [2] EncryptedData -- EncAPRepPart
}

EncAPRepPart    ::= [APPLICATION 27] SEQUENCE {
        ctime           [0] KerberosTime,
        cusec           [1] Microseconds,
        subkey          [2] EncryptionKey OPTIONAL,
        seq-number      [3] UInt32 OPTIONAL
}

KRB-SAFE        ::= [APPLICATION 20] SEQUENCE {
        pvno            [0] INTEGER (5),
        msg-type        [1] INTEGER (20),
        safe-body       [2] KRB-SAFE-BODY,
        cksum           [3] Checksum
}

KRB-SAFE-BODY   ::= SEQUENCE {
        user-data       [0] OCTET STRING,
        timestamp       [1] KerberosTime OPTIONAL,
        usec            [2] Microseconds OPTIONAL,
        seq-number      [3] UInt32 OPTIONAL,
        s-address       [4] HostAddress,
        r-address       [5] HostAddress OPTIONAL
}

KRB-PRIV        ::= [APPLICATION 21] SEQUENCE {
        pvno            [0] INTEGER (5),
        msg-type        [1] INTEGER (21),
                        -- NOTE: there is no [2] tag
```

```
        enc-part        [3] EncryptedData -- EncKrbPrivPart
}

EncKrbPrivPart  ::= [APPLICATION 28] SEQUENCE {
        user-data       [0] OCTET STRING,
        timestamp       [1] KerberosTime OPTIONAL,
        usec            [2] Microseconds OPTIONAL,
        seq-number      [3] UInt32 OPTIONAL,
        s-address       [4] HostAddress -- sender's addr --,
        r-address       [5] HostAddress OPTIONAL -- recip's addr
}

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

EncKrbCredPart  ::= [APPLICATION 29] SEQUENCE {
        ticket-info     [0] SEQUENCE OF KrbCredInfo,
        nonce           [1] UInt32 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
}

KRB-ERROR       ::= [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] Int32,
        crealm          [7] Realm OPTIONAL,
        cname           [8] PrincipalName OPTIONAL,
        realm           [9] Realm -- service realm --,
        sname           [10] PrincipalName -- service name --,
        e-text          [11] KerberosString OPTIONAL,
        e-data          [12] OCTET STRING OPTIONAL
}

METHOD-DATA     ::= SEQUENCE OF PA-DATA

TYPED-DATA      ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
        data-type       [0] Int32,
        data-value      [1] OCTET STRING OPTIONAL
}

-- preauth stuff follows

PA-ENC-TIMESTAMP        ::= EncryptedData -- PA-ENC-TS-ENC

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

ETYPE-INFO-ENTRY        ::= SEQUENCE {
        etype           [0] Int32,
        salt            [1] OCTET STRING OPTIONAL
}

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

ETYPE-INFO2-ENTRY       ::= SEQUENCE {
        etype           [0] Int32,
        salt            [1] KerberosString OPTIONAL,
        s2kparams       [2] OCTET STRING OPTIONAL
}

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

AD-IF-RELEVANT          ::= AuthorizationData

AD-KDCIssued            ::= SEQUENCE {
        ad-checksum     [0] Checksum,
        i-realm         [1] Realm OPTIONAL,
        i-sname         [2] PrincipalName OPTIONAL,
        elements        [3] AuthorizationData
```

```
}

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

AD-MANDATORY-FOR-KDC    ::= AuthorizationData

END
```

B. Changes since RFC 1510

This document replaces RFC 1510 and clarifies specification of items that were not completely specified. Where changes to recommended implementation choices were made, or where new options were added, those changes are described within the document and listed in this section. More significantly, "Specification 2" in Section 8 changes the required encryption and checksum methods to bring them in line with the best current practices and to deprecate methods that are no longer considered sufficiently strong.

Discussion was added to Section 1 regarding the ability to rely on the KDC to check the transited field, and on the inclusion of a flag in a ticket indicating that this check has occurred. This is a new capability not present in RFC 1510. Pre-existing implementations may ignore or not set this flag without negative security implications.

The definition of the secret key says that in the case of a user the key may be derived from a password. In RFC 1510, it said that the key was derived from the password. This change was made to accommodate situations where the user key might be stored on a smart-card, or otherwise obtained independently of a password.

The introduction mentions the use of public key cryptography for initial authentication in Kerberos by reference. RFC 1510 did not include such a reference.

Section 1.3 was added to explain that while Kerberos provides authentication of a named principal, it is still the responsibility of the application to ensure that the authenticated name is the entity with which the application wishes to communicate.

Discussion of extensibility has been added to the introduction.

Discussion of how extensibility affects ticket flags and KDC options was added to the introduction of Section 2. No changes were made to existing options and flags specified in RFC 1510, though some of the

sections in the specification were renumbered, and text was revised to make the description and intent of existing options clearer, especially with respect to the ENC-TKT-IN-SKEY option (now section 2.9.2) which is used for user-to-user authentication. The new option and ticket flag transited policy checking (Section 2.7) was added.

A warning regarding generation of session keys for application use was added to Section 3, urging the inclusion of key entropy from the KDC generated session key in the ticket. An example regarding use of the sub-session key was added to Section 3.2.6. Descriptions of the pa-etype-info, pa-etype-info2, and pa-pw-salt pre-authentication data items were added. The recommendation for use of pre-authentication was changed from "MAY" to "SHOULD" and a note was added regarding known plaintext attacks.

In RFC 1510, Section 4 described the database in the KDC. This discussion was not necessary for interoperability and unnecessarily constrained implementation. The old Section 4 was removed.

The current Section 4 was formerly Section 6 on encryption and checksum specifications. The major part of this section was brought up to date to support new encryption methods, and moved to a separate document. Those few remaining aspects of the encryption and checksum specification specific to Kerberos are now specified in Section 4.

Significant changes were made to the layout of Section 5 to clarify the correct behavior for optional fields. Many of these changes were made necessary because of improper ASN.1 description in the original Kerberos specification which left the correct behavior underspecified. Additionally, the wording in this section was tightened wherever possible to ensure that implementations conforming to this specification will be extensible with the addition of new fields in future specifications.

Text was added describing time_t=0 issues in the ASN.1. Text was also added, clarifying issues with implementations treating omitted optional integers as zero. Text was added clarifying behavior for optional SEQUENCE or SEQUENCE OF that may be empty. Discussion was added regarding sequence numbers and behavior of some implementations, including "zero" behavior and negative numbers. A compatibility note was added regarding the unconditional sending of EncTGSRepPart regardless of the enclosing reply type. Minor changes were made to the description of the HostAddresses type. Integer types were constrained. KerberosString was defined as a (significantly) constrained GeneralString. KerberosFlags was defined to reflect existing implementation behavior that departs from the

definition in RFC 1510. The transited-policy-checked(12) and the ok-as-delegate(13) ticket flags were added. The disable-transited-check(26) KDC option was added.

Descriptions of commonly implemented PA-DATA were added to Section 5. The description of KRB-SAFE has been updated to note the existing implementation behavior of double-encoding.

There were two definitions of METHOD-DATA in RFC 1510. The second one, intended for use with KRB_AP_ERR_METHOD was removed leaving the SEQUENCE OF PA-DATA definition.

Section 7, naming constraints, from RFC 1510 was moved to Section 6.

Words were added describing the convention that domain-based realm names for newly-created realms should be specified as uppercase. This recommendation does not make lowercase realm names illegal. Words were added highlighting that the slash-separated components in the X.500 style of realm names is consistent with existing RFC 1510 based implementations, but that it conflicts with the general recommendation of X.500 name representation specified in RFC 2253.

Section 8, network transport, constants and defined values, from RFC 1510 was moved to Section 7. Since RFC 1510, the definition of the TCP transport for Kerberos messages was added, and the encryption and checksum number assignments have been moved into a separate document.

"Specification 2" in Section 8 of the current document changes the required encryption and checksum methods to bring them in line with the best current practices and to deprecate methods that are no longer considered sufficiently strong.

Two new sections, on IANA considerations and security considerations were added.

The pseudo-code has been removed from the appendix. The pseudo-code was sometimes misinterpreted to limit implementation choices and in RFC 1510, it was not always consistent with the words in the specification. Effort was made to clear up any ambiguities in the specification, rather than to rely on the pseudo-code.

An appendix was added containing the complete ASN.1 module drawn from the discussion in Section 5 of the current document.

END NOTES

(*TM) Project Athena, Athena, and Kerberos are trademarks of the Massachusetts Institute of Technology (MIT).

Normative References

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005.

[ISO-646/ECMA-6] International Organization for Standardization, "7-bit Coded Character Set for Information Interchange", ISO/IEC 646:1991.

[ISO-2022/ECMA-35] International Organization for Standardization, "Character code structure and extension techniques", ISO/IEC 2022:1994.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC2253] Wahl, M., Kille, S., and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[RFC3513] Hinden, R. and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

[X680] Abstract Syntax Notation One (ASN.1): Specification of Basic Notation, ITU-T Recommendation X.680 (1997) | ISO/IEC International Standard 8824-1:1998.

[X690] ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), ITU-T Recommendation X.690 (1997)| ISO/IEC International Standard 8825-1:1998.

Informative References

[ISO-8859] International Organization for Standardization, "8-bit Single-byte Coded Graphic Character Sets -- Latin Alphabet", ISO/IEC 8859.

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

[DGT96] Don Davis, Daniel Geer, and Theodore Ts'o, "Kerberos With Clocks Adrift: History, Protocols, and Implementation", USENIX Computing Systems 9:1, January 1996.

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

[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.

[MNSS87] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, Section E.2.1: Kerberos Authentication and Authorization System, M.I.T. Project Athena, Cambridge, Massachusetts, December 21, 1987.

[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.

[Neu93] B. Clifford Neuman, "Proxy-Based Authorization and Accounting for Distributed Systems," in Proceedings of the 13th International Conference on Distributed Computing Systems, Pittsburgh, PA, May 1993.

[NT94] B. Clifford Neuman and Theodore Y. Ts'o, "An Authentication Service for Computer Networks," IEEE Communications Magazine, Vol. 32 (9), p. 33-38, September 1994.

[Pat92] J. Pato, Using Pre-Authentication to Avoid Password Guessing Attacks, Open Software Foundation DCE Request for Comments 26 (December 1992.

[RFC1510] Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[RFC4086] Eastlake, D., 3rd, Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[SNS88] J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," p. 191-202, Usenix Conference Proceedings, Dallas, Texas, February 1988.

[RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005.

Authors' Addresses

Clifford Neuman
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, USA

EMail: bcn@isi.edu

Tom Yu
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139, USA

EMail: tlyu@mit.edu

Sam Hartman
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139, USA

EMail: hartmans-ietf@mit.edu

Kenneth Raeburn
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139, USA

EMail: raeburn@mit.edu

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.