Schema for the Kerberos V5 KDC Server

Document Version: 7

Date: 10/20/00

For questions or comments concerning this document, please send an email note to dce-ldap@opengroup.org or call Donna Skibbie at (512) 838-3896.

1. Summary of Changes

The following is a summary of the changes that were made since Version 6 of this document:

- A note has been added about redundant attributes in KrbPolicy
- The following new attributes have been added:
  - keyExpires
  - krb-Attributes
  - krbCreatorsName
  - krbCreateTimestamp
  - krbModifiersName
  - krbModififyTimestamp
  - multKeyVersionsOK
  - nextKeyVersion
  - krbAliases
- The following attributes from the Netscape schema have been added:
  - passwordDictFiles
  - passwordMaxAge
  - passwordMinAge
  - passwordMinDiffChars
  - passwordMinLength
- The following attribute from the Tivoli/IBM Schema has been added
  - secAcctExpires
- The EQUALITY has been changed to caseExactMatch on the following attributes:
  - krbPrincipalName
  - krbRealmName
- A new encryption type (ENCTYPE_RSA_PRIVKEY) has been added to the encType attribute.
- The following attributes have been replaced
  - logType, replaced by cn=KrbLog
  - minPwdClasses, replaced by passwordMinDiffChars
  - admDictDB, replaced by passwordDictFiles
  - curKeyVersion, replaced by curKeyVersionArray
- KrbAlias has been changed to an auxiliary class
- KrbKey contains new MUST and MAY
- KrbLog contains cn instead of logType
- KrbPolicy contains new attributes, some of which are redundant
- KrbPrincipal contains new attributes
- KrbRealmExt contains some different attributes

2. Open Issues to Discuss

- Should we change KrbRealm to an auxiliary object class so it can be attached to a domain entry?
- Should we add a prefix (such as "krb-") to the names of all attributes and object classes defined in this schema to satisfy registration requirements of Active Directory?

3. Introduction

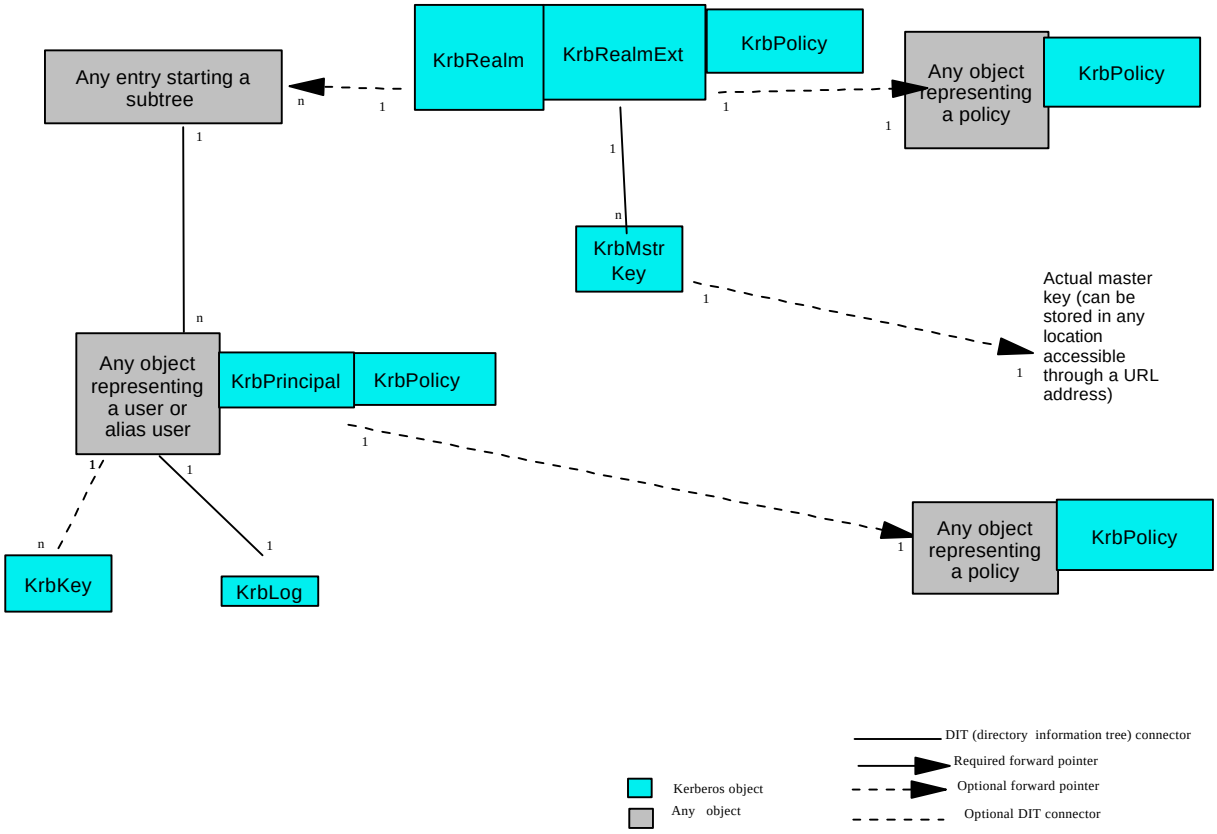
This document defines an LDAP schema for storing attributes used by the MIT implementation of the Kerberos Version 5 Key Distribution (KDC) Server. These attributes include:

- Attributes that define a Kerberos realm--These attributes map to attributes defined in the KDC.conf file of the MIT implementation.
- Attributes that define principals in a Kerberos realm--These attributes map to attributes defined in the principal principal and administration databases of the MIT implementation.

4. Overview

The Kerberos Version 5 KDC LDAP schema is designed to meet four objectives. The first objective is to use LDAP attributes already defined by standards organizations. The second objective is to use attributes already defined by existing LDAP implementations that store that store Kerberos Version 5 KDC attributes. The third objective is to provide a way of sharing common security attributes, such as password policy attributes, with non-Kerberos applications. The fourth objective is to provide a way of protecting keys and other sensitive information.

The following figure illustrates the schema:



5. Configuration

This section provides a high-level description of the tasks that need to be performed to configure this schema. This description is provided to help understand the schema. The description assumes the administrator is using standard LDAP Version 3 interfaces to do the configuration tasks unless noted to the contrary. In practice, a tool could be provided to the administrator that would automate many of these configuration tasks.

5.1 Configuring the Realm Portion of the Schema

The administrator must do the following:

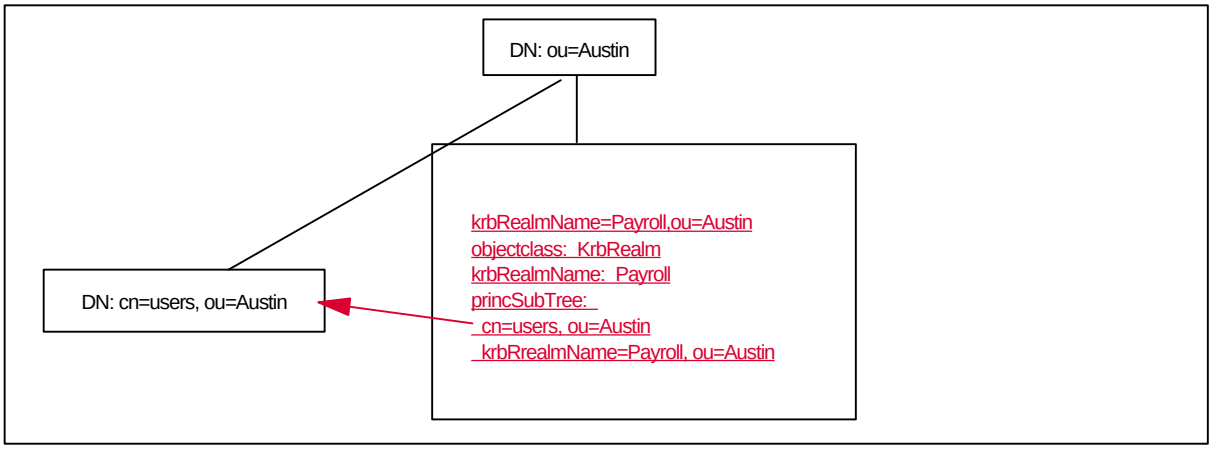
- Configure a realm entry
- Add KrbRealmExt to the realm entry
- Add KrbPolicy to the realm entry or a referenced entry
- Configure one or more master key (KrbMstrKey) entries

5.1.1 Configuring a Realm Entry

The administrator must configure a realm entry. The administrator does this by creating an entry anywhere in the directory with a structural object class of KrbRealm and configuring the following attributes:

- krbRealmName—The name of the realm
- princSubtree—The DN of each sub-tree in the directory under which principals in the realm will reside.

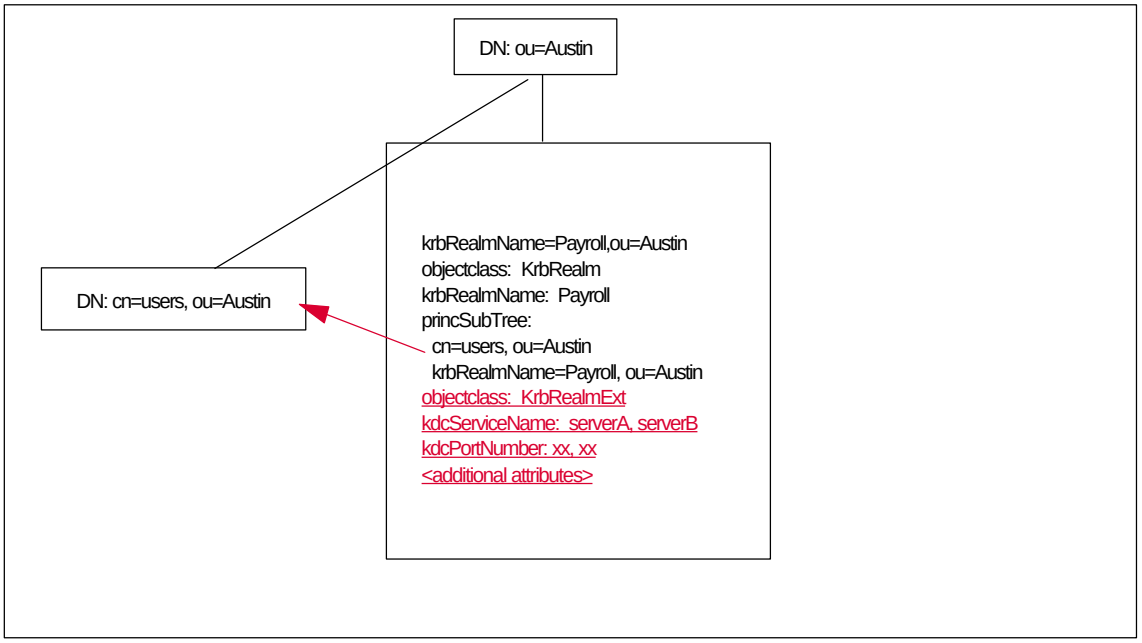
In the following example, the realm entry is “krbRealmName=Payroll, ou=Austin.” The name of the realm is Payroll. The principals in the realm will reside under two subtrees: "ou=Users, ou=Austin" and the realm entry (“krbRealmName=Payroll, ou=Austin”).



5.1.2 Adding KrbRealmExt to the Realm Entry

The administrator must add the KrbRealmExt auxiliary object class to the realm entry and configure the attributes in KrbRealmExt. Th KrbRealmExt attributes provide additional

information about the realm such as the KDC servers in the realm, the port numbers used by the KDC servers, and so forth. The following figure is an example:

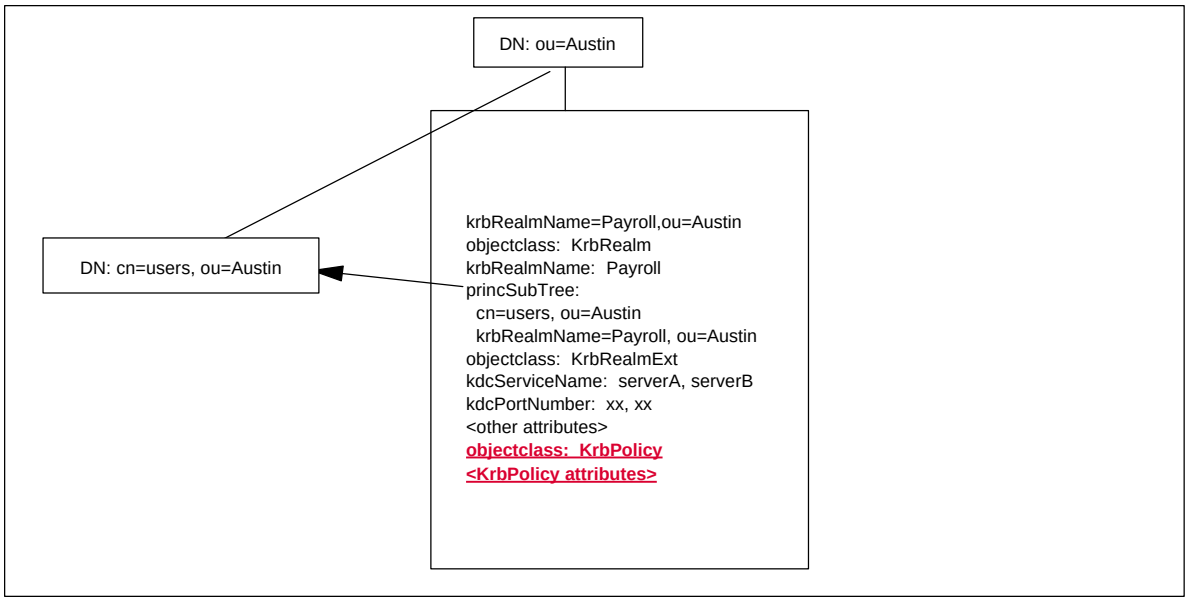


5.1.3 Adding KrbPolicy to the Realm Entry or a Policy Entry

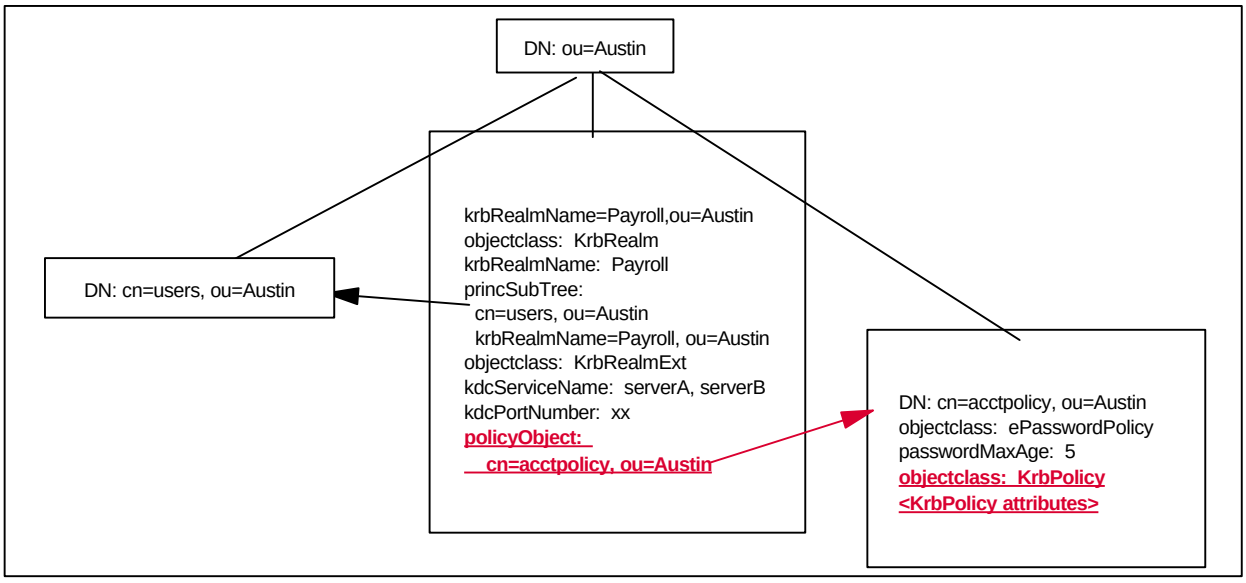
The administrator must add the KrbPolicy auxiliary class and its attributes to the realm entry or a referenced policy entry. The referenced policy entry can reside anywhere in the directory and can be any type entry. The administrator might use a referenced policy entry for either or both of the following reasons:

- to allow a realm to share realm policy attributes with other realms.
- to allow a realm to share generic policy attributes (such as password policy attributes) with non-Kerberos applications.

In the following example, the administrator configured the realm policy in the realm entry.



In the following example, the administrator configured the realm policy in a referenced policy entry.



5.1.4 A Note about Redundant Attributes in KrbPolicy

The KrbPolicy auxiliary object class contains some redundant attributes; that is two attributes with the same meaning. For example, the maxPwdAge attribute has the same meaning as the passwordMaxAge attribute. The reason for this redundancy is that the Microsoft, IBM/Tivoli, and Netscape schemas have defined some different attributes for storing the same information. By including attribute definitions from all three schemas in KrbPolicy, the administrator has a choice of which definition to use.

In the previous example, the maxPwdAge attribute is defined in the Microsoft Active Directory schema and the passwordMaxAge attribute is defined in the IBM/Tivoli and Netscape schemas. If the administrator is in a Microsoft environment, the administrator might want to use the maxPwdAge attribute, because other Microsoft-based applications might be configured to use the same attribute. If the administrator is in an IBM, Tivoli, or Netscape environment, the administrator might want to use the passwordMaxAge attribute, because other IBM, Tivoli, or Netscape applications might be configured to use this attribute.

The following table lists the redundant attributes and where these attributes are defined:

Attribute 1	Where Defined	Attribute 2	Where Defined
accountExpires	Microsoft Active Directory shema	secAcctLife	Tivoli/IBM schema
maxPwdAge	Microsoft Active Directory schema	passwordMaxAge	Netscape and Tivoli/IBM schemas
minPwdAge	Microsoft Active Directory schema	passwordMinAge	Netscape and Tivoli/IBM schemas
minPwdLength	Microsoft Active Directory schema	passwordMinLength	Netscape and Tivoli/IBM schemas
no attribute (password expiration date computed from the pwdLastSet and maxPwdAge attributes)	Microsoft Active Directory schema	passwordExpireTime	Netscape and Tivoli/IBM schemas

Implementations of Kerberos need to check for redundant attributes in an entry. If a contradiction occurs between two redundant attributes, Kerberos needs to use the attribute with the more restrictive value. For example, if an entry contains maxPwdAge=5 and passwordMaxAge=10, Kerberos needs to use maxPwdAge because this attribute has the more restrictive value for maximum password age.

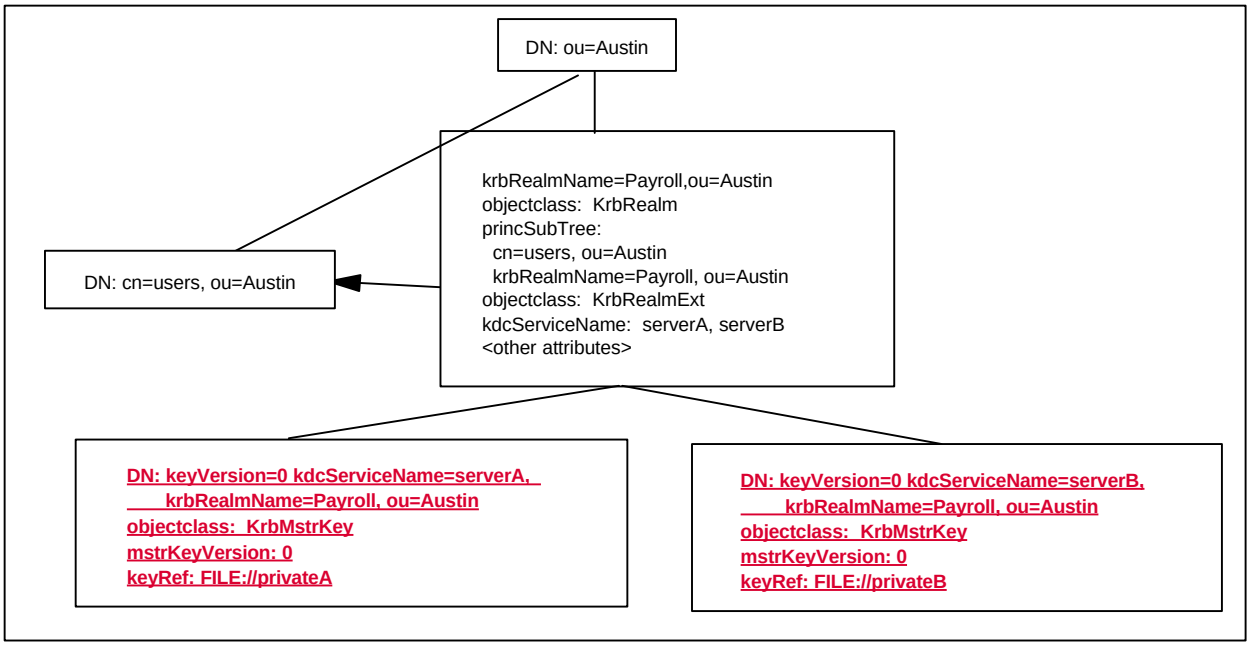
5.1.5 Configuring Master Key (KrbMstrKey) Entries

The administrator must configure one or more master key entries. Each master key entry represents a master key that will be used by one or more KDC servers in the realm. The administrator can configure a single master key entry for all the KDC servers in the realm, or one master key entry for each KDC server in the realm.

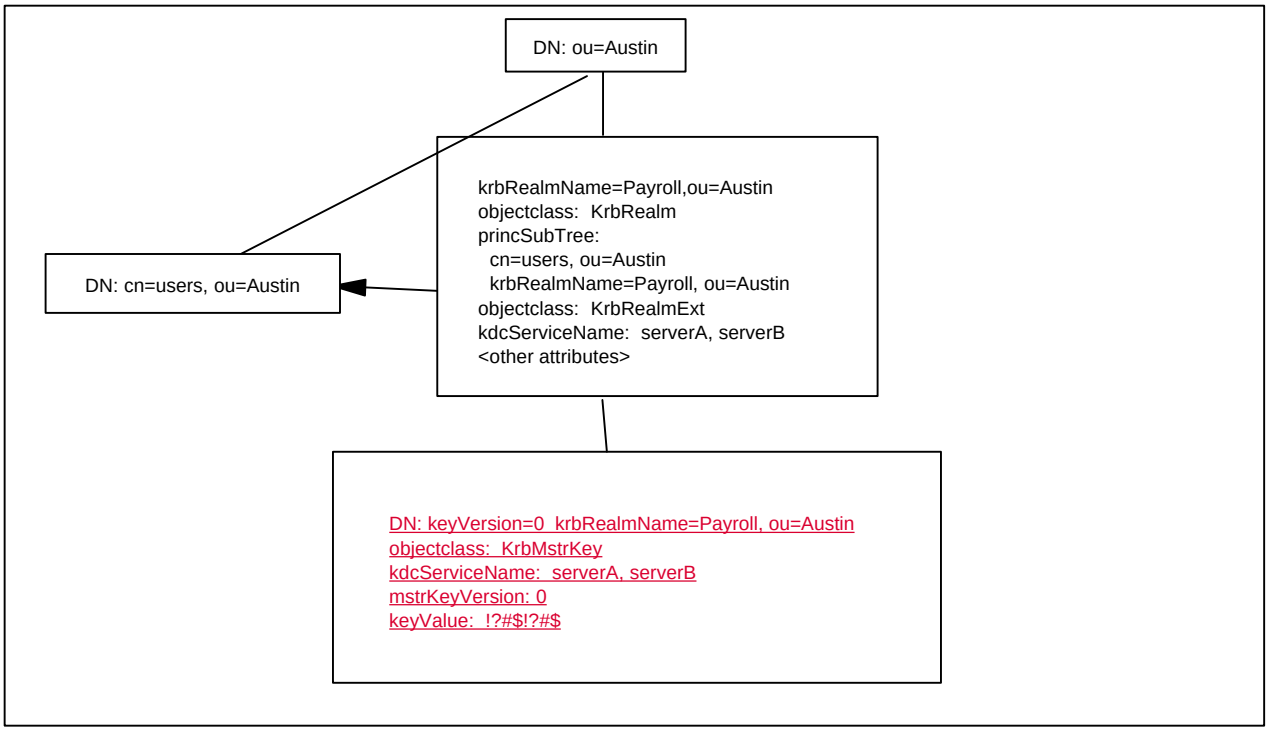
To configure a master key entry, the administrator creates a new entry under the realm entry that is of object class KrbMstrKey and that contains the required KrbMstrKey attributes. These attributes include:

- the version of the master key (keyVersion)
- the master key value (keyValue) or the location where the master key is stored (keyRef)
- The KDC servers that will use this master key (kdcServiceName)

In the following example, the administrator configured two master keys. One belongs to KDC server A. The other belongs to KDC server B. Each master key is stored in a private file on the KDC server.



In the following example, the administrator configured one master key entry. The entry is shared by KDC Servers A and B, and the master key is stored in the LDAP entry.



5.2 Configuring the Principal Portion of the Schema

The administrator needs to do the following for each principal that will reside in the realm:

- Determine a principal name
- Configure the principal entry
- Add KrbPolicy to the principal entry or to a referenced policy entry
- Configure a password for the principal entry

During runtime, the KDC will create a KrbLog entry for the principal entry and update this entry with login activity related to the principal.

5.2.1 Determining a Principal Name

The administrator must select a Kerberos name for the principal. The name must be in the format principal@realm and must be unique within the realm. To verify that a name is unique, the administrator must search each subtree listed in the princSubtree attribute of the realm entry for an entry where krbPrincipalName = principal@realm. If no such entry is found, the name is unique and can be used.

For example, assume the Payroll realm administrator wants to assign Mary Smith the name marys@Payroll. Before doing this, the administrator must verify that this name is unique within the Payroll realm. The administrator does this by searching all entries under “cn=Users, ou=Austin” and the “krbRealmName=Payroll, ou=Austin” for an entry where krbPrincipalName = marys@payroll. If no match is found, the name is unique within the realm.

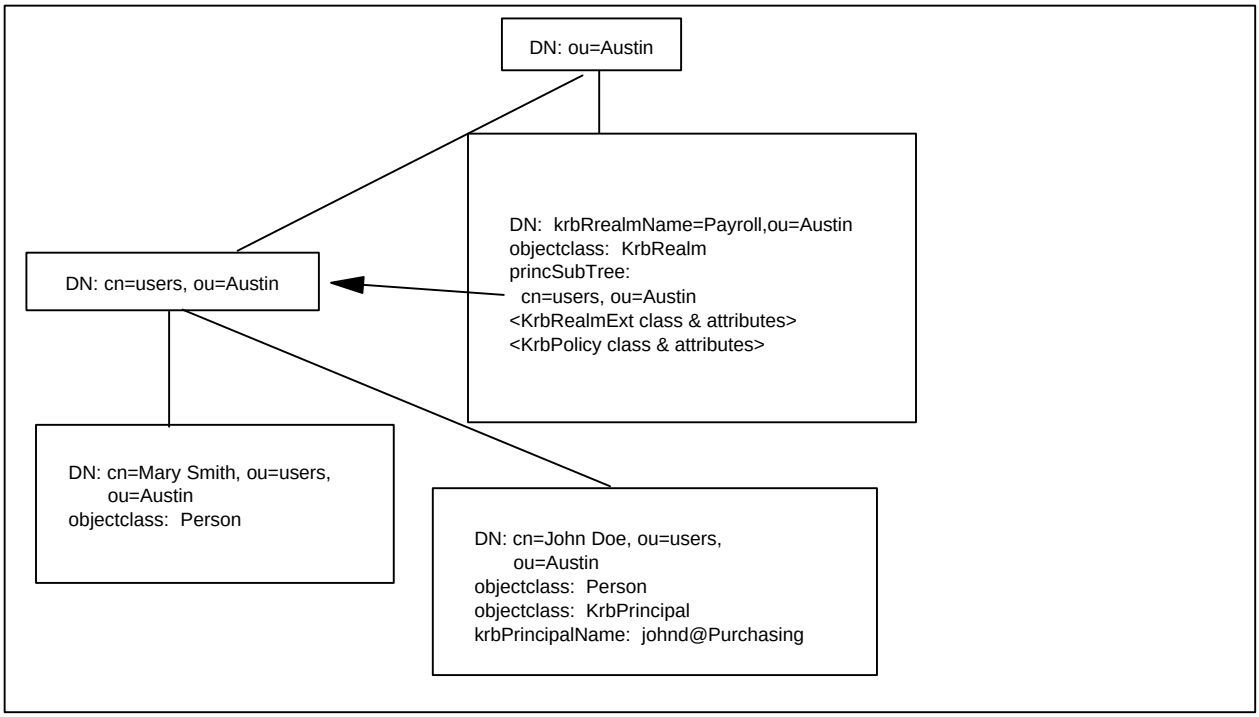
5.2.2 Configuring a Principal Entry

The administrator must determine which entry in the directory will represent the principal. This entry is called the “principal entry.” The principal entry can be an existing directory entry or a new entry created by the administrator. It can be of any structural object class (such as person, iphost, or alias). The only provisions are that the entry must:

- reside under one of the subtrees listed in the princSubtree attribute of the realm entry
- not already be configured with the krbPrincipalName attribute

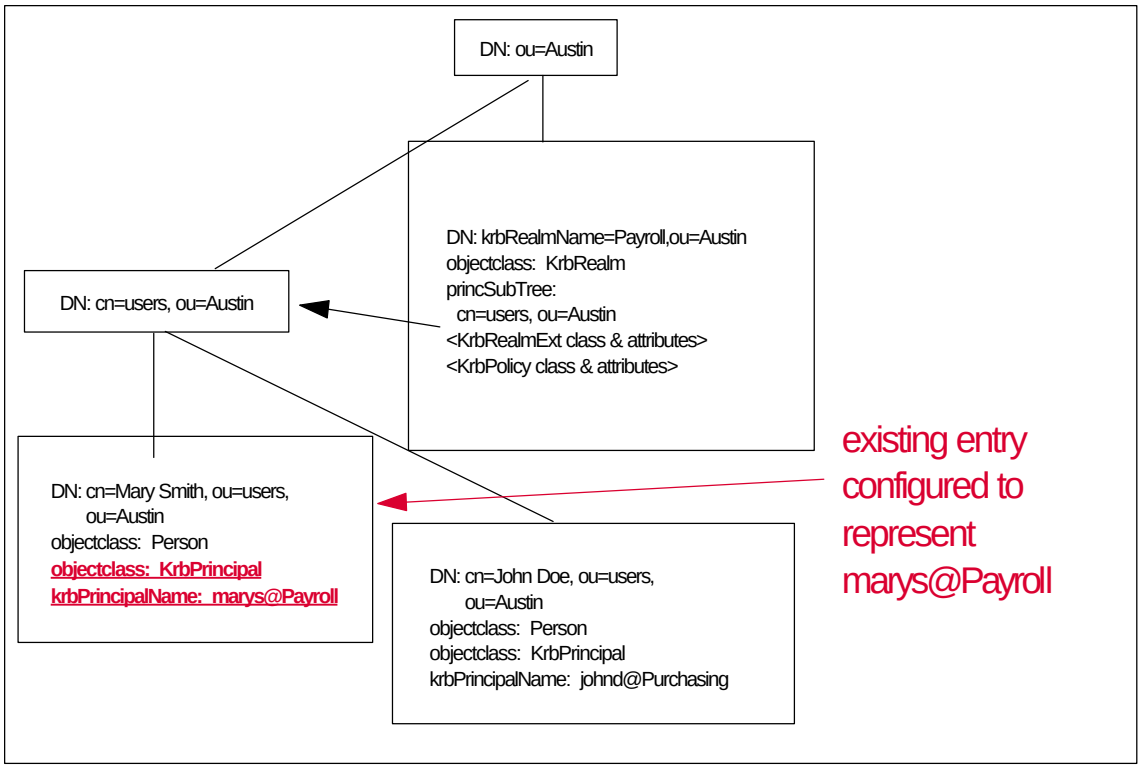
After determining the principal entry, the administrator must add the KrbPrincipal auxiliary object class to the principal entry and configure the krbPrincipalName attribute with the principal name.

The following figure is an example of an existing configuration. This figure will be referenced in the examples that follow.



5.2.2.1 Example 1: Configuring an Existing Entry to Represent a Principal

Assume the administrator needs to configure Mary Smith as a principal in the Payroll realm with the principal name of marys@Payroll. An entry named “cn=Mary Smith, ou=users, ou=Austin” already exists. This entry represents the Mary Smith person. The administrator can use this existing entry to represent the marys@Payroll principal because the existing entry meets the two conditions described previously. (The existing entry resides under the “cn=users, ou=Austin,” which is one of the subtrees listed in the princSubtree attribute of the Payroll realm, and the entry is not already configured with the krbPrincipalName attribute.) Therefore, the administrator uses the existing entry to represent marys@Payroll principal. The administrator does this by adding the KrbPrincipal auxiliary class to the existing entry and configuring krbPrincipalName with marys@Payroll.



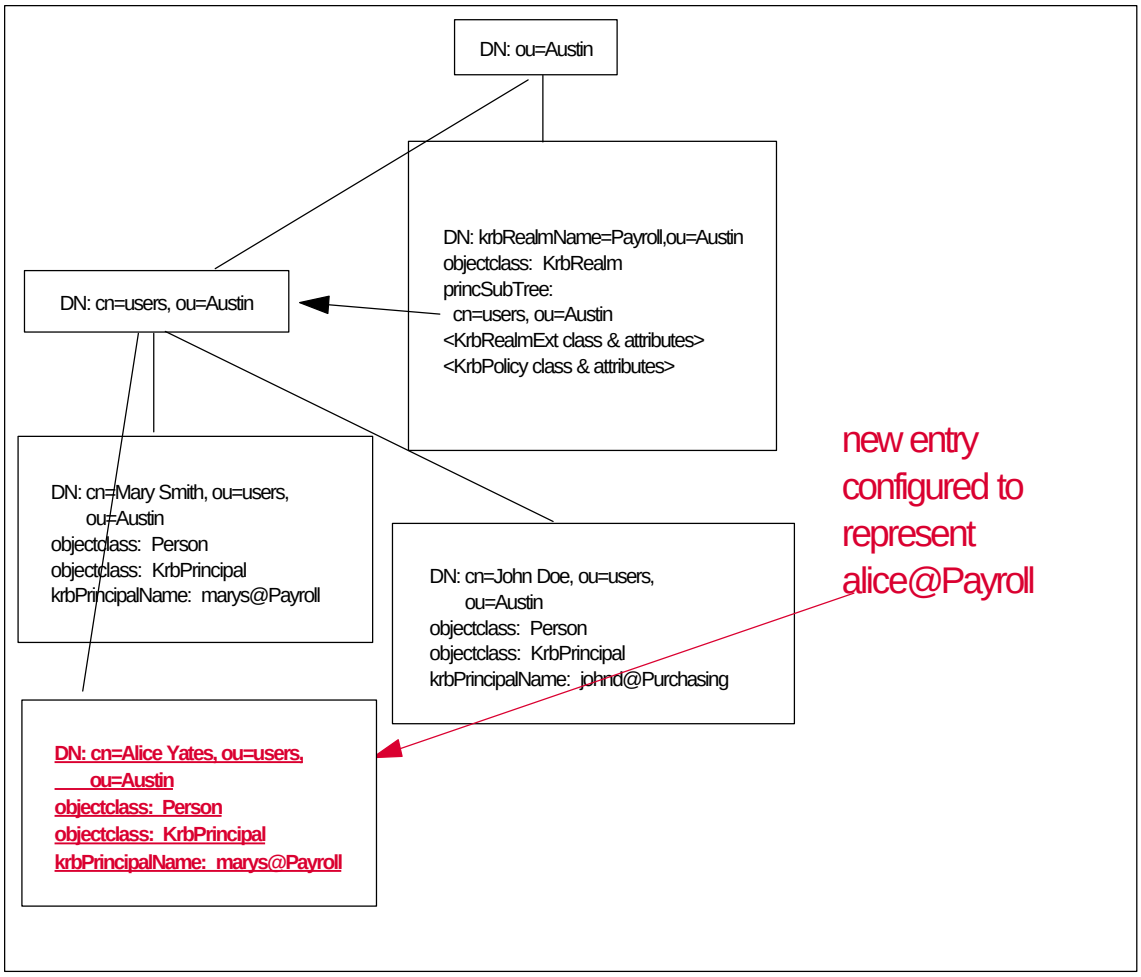
5.2.2.2 Example 2: Using an New Entry to Represent a Principal

Assume the administrator needs to configure Alice Yates as a principal in the Payroll realm with the principal name of alice@Payroll. No entry currently exists for Alice Yates. Therefore, the administrator must create a new entry for Alice Yates and configure this entry to be a principal in the Payroll realm.

The administrator can create the new entry using any structural object class and configure the attributes of this structural object class in any way desired. In this example, the administrator uses the structural object class of Person and configures the cn attribute of Person with “Alice Yates.”

The administrator configures the DN of the new entry to be “Alice Yates, ou=users, ou=Austin.” This DN is acceptable because it locates the entry under one of the subtrees configured for the Payroll realm (“ou=users, ou=Austin”). The administrator configures the new entry to represent alice@Payroll principal by adding the KrbPrincipal auxiliary class to the new entry and configuring krbPrincipalName with alice@Payroll.

The following figure illustrates the new entry.



5.2.2.3 Example 3: Using a KrbAlias Entry to Represent a Principal

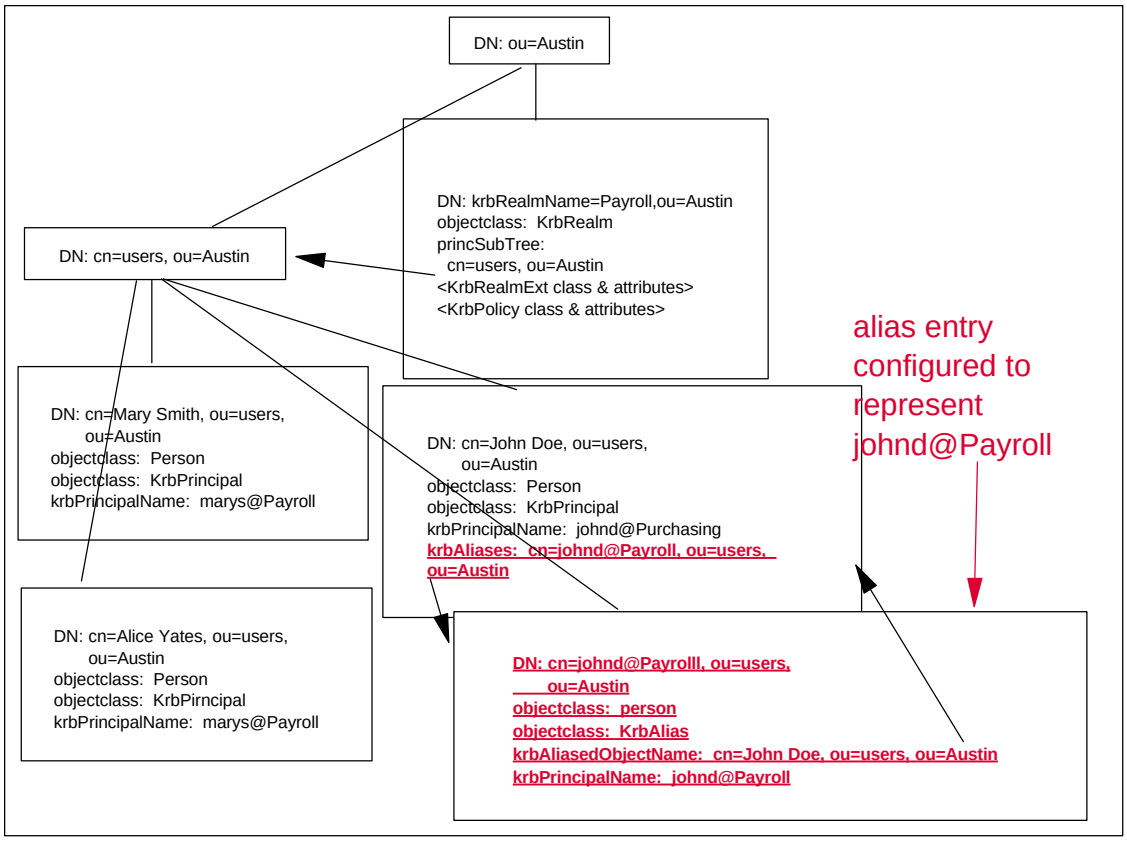
Assume the administrator needs to configure John Doe as a principal in the Payroll realm with the principal name of johnd@Payroll. An entry already exists that represents the John Doe person. However, this entry does not meet the two requirements previously described, because it already contains a krbPrincipalName attribute.

Therefore, the administrator must configure a new entry to represent johnd@Payroll. The administrator can do this in a similar way as described in Example 2. However, in this example, the administrator might want to configure the new entry as a Kerberos alias to the John Doe entry. The administrator does this by:

1. Attaching the KrbAlias auxiliary object class to the new entry and using the krbAliasNameObject to associate the new entry with the John Doe person entry.
2. Adding the HintKrbAliases attribute to the John Doe entry and using the HintKrbAliases attribute to list the new entry as an alias for the John Doe entry.

Kerberos will ignore the krbAliasNameObject and HintKrbAliases attributes. However, higher level applications can use these attributes to associate the johnd@Payroll entry with the John Doe person entry.

The following figure illustrates this configuration.

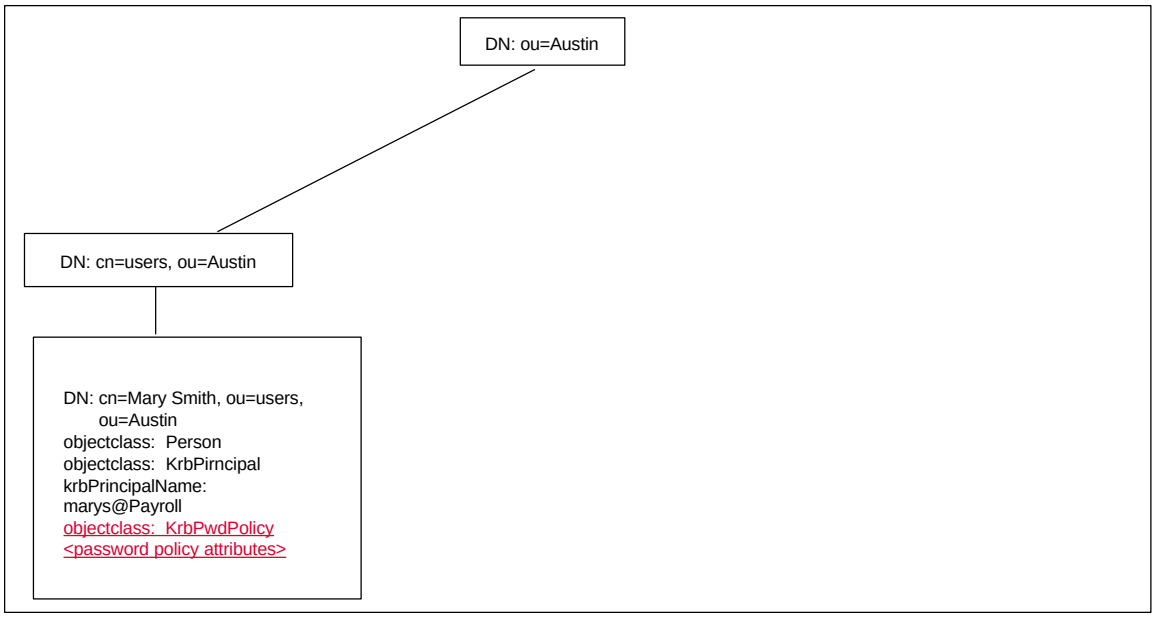


5.2.3 Adding KrbPolicy to the Principal Entry or a Policy Entry

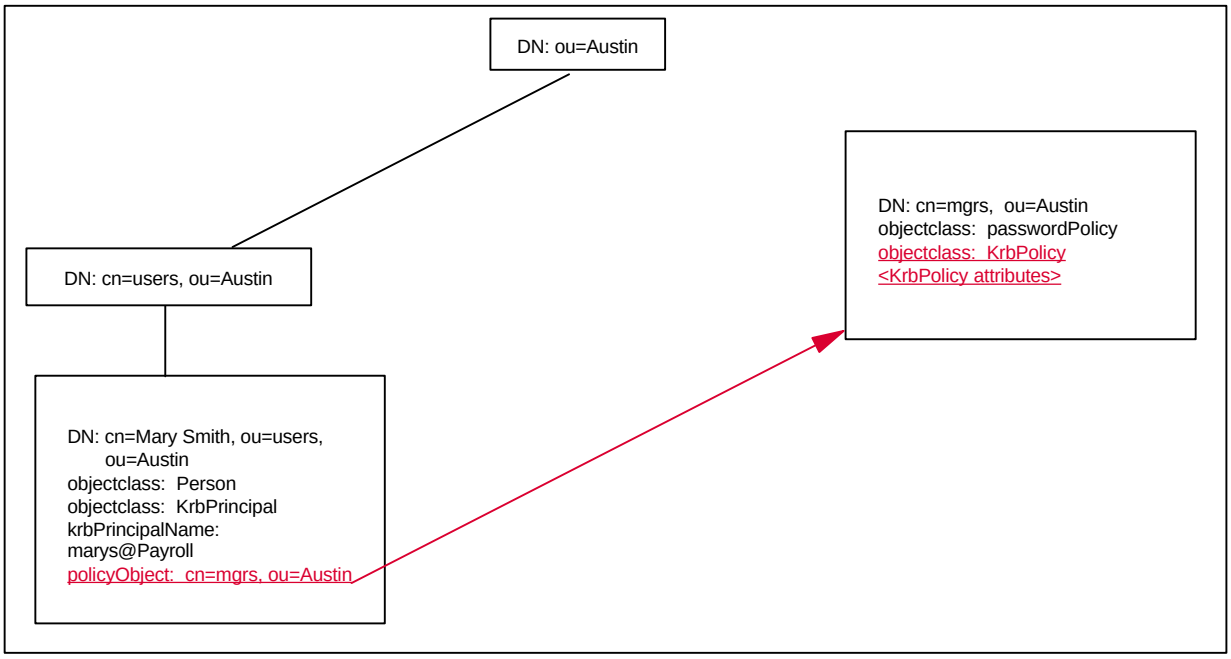
The administrator must add the KrbPolicy auxiliary class and its attributes to either the entry representing the principal or a referenced policy entry. The referenced policy entry can reside anywhere in the directory and can be any type entry. The administrator might use a referenced policy entry for one or both of the following reasons:

- to allow a principal to share policy attributes with other principals
- to allow a principal to share generic policy attributes (such as password policy attributes) with non-Kerberos applications)

The following figure is an example in which KrbPolicy is configured in the entry representing the principal:



The following is an example in which KrbPolicy is configured in a referenced policy entry.



Note: The KrbPolicy auxiliary object class contains some redundant attributes. This is explained in the previous section "A Note About Redundant Attributes in KrbPolicy."

5.2.4 Configuring a Password for the Principal Entry

The user represented by the principal entry must configure a password for the principal. The way that the user does this depends on how the policy for the principal is configured.

5.2.4.1 Default Policy for Configuring a Password

In the default policy, the KDC requires the password for a principal to be stored in one or more KrbKey entries that only the KDC servers in the realm can understand. Therefore, the user must configure a password for a principal using either a Kerberos administration tool or a trusted administration tool.

A trusted administration tool is a tool trusted by the KDC to create a KrbKey entry. If the administrator wants to allow users to configure passwords using a trusted administration tool, the administrator configures the DN name of this tool in the realm entry. The administrator also must provide a way for the trusted administration tool to access the master keys configured in the realm.

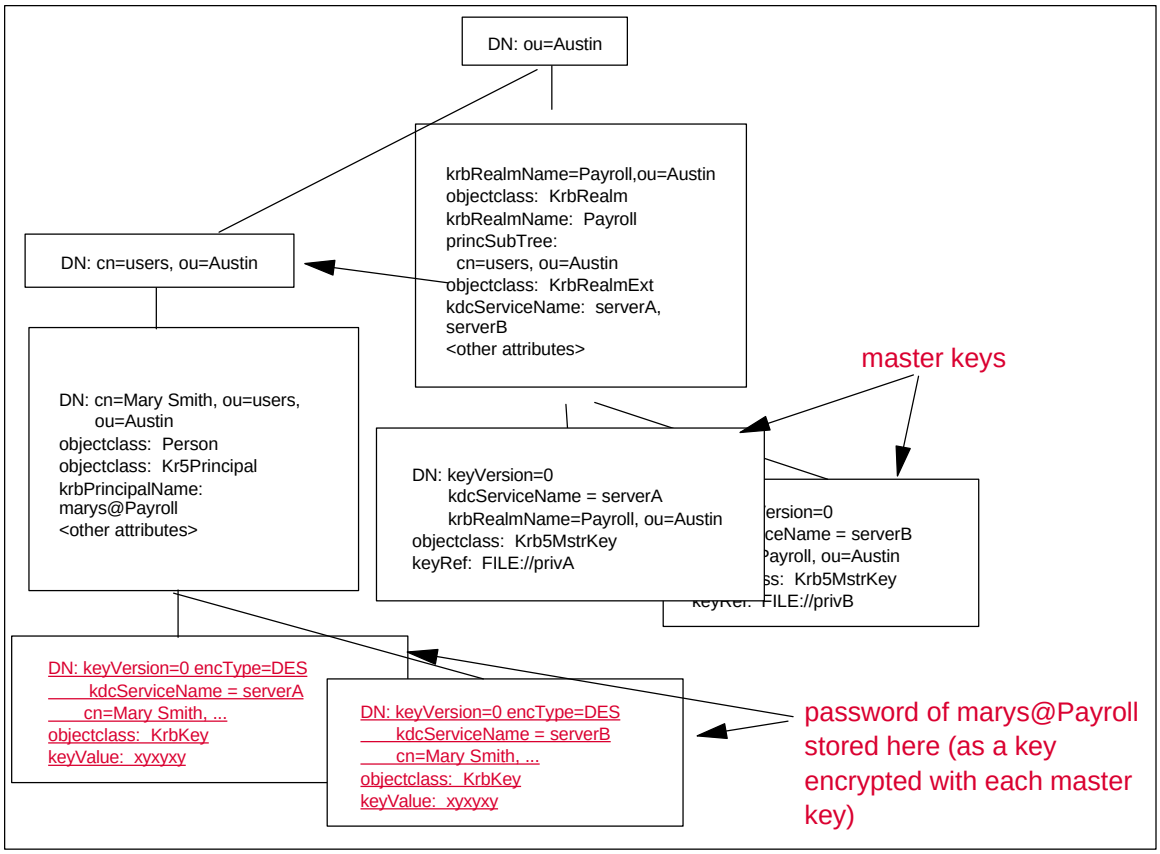
The Kerberos administration tool or the trusted administration tool must create the KrbKey entry as follows:

1. Get the password from the user. The user can supply the password in plaintext form or as a key.
2. If the password is plaintext, get the salt from the user, if supplied. If the user does not supply a salt, generate a salt using the mechanism implied by a salttype supplied by the user. If the user does not supply a salttype, generate a salt using a default salttype mechanism (based on the principal and realm names). Then, transform the plaintext password into a key, using the salt to seed the transformation algorithm.
3. Encrypt the key with the master key.
4. Create a KrbKey entry under the principal entry and, in this entry, store the encrypted key and information used to produce the encrypted key.

If multiple master keys are configured in the realm, the Kerberos or trusted administration tool must create one KrbKey entry for each master key.

For example, assume Mary Smith from the previous example wants to configure a password to use with her marys@Payroll principal identity. She must use a Kerberos administration tool or a trusted administration tool to configure this password.

The following figure illustrates the KrbKey entries created by the administration tool. The realm is configured with two master keys, so the administration tool created two KrbKey entries for the same password. One KrbKey entry contains a DES key transformation of the password encrypted with the master key of KDC ServerA. The other KrbKey entry contains a DES key transformation of the password encrypted in the master key of KDC ServerB. When the administration tool created the two KrbKey entries, it set the ACLs on these two entries so that only a KDC server in the realm can read them.



5.2.4.2 Alternate Method for Configuring a Password (USE_USER_PWD)

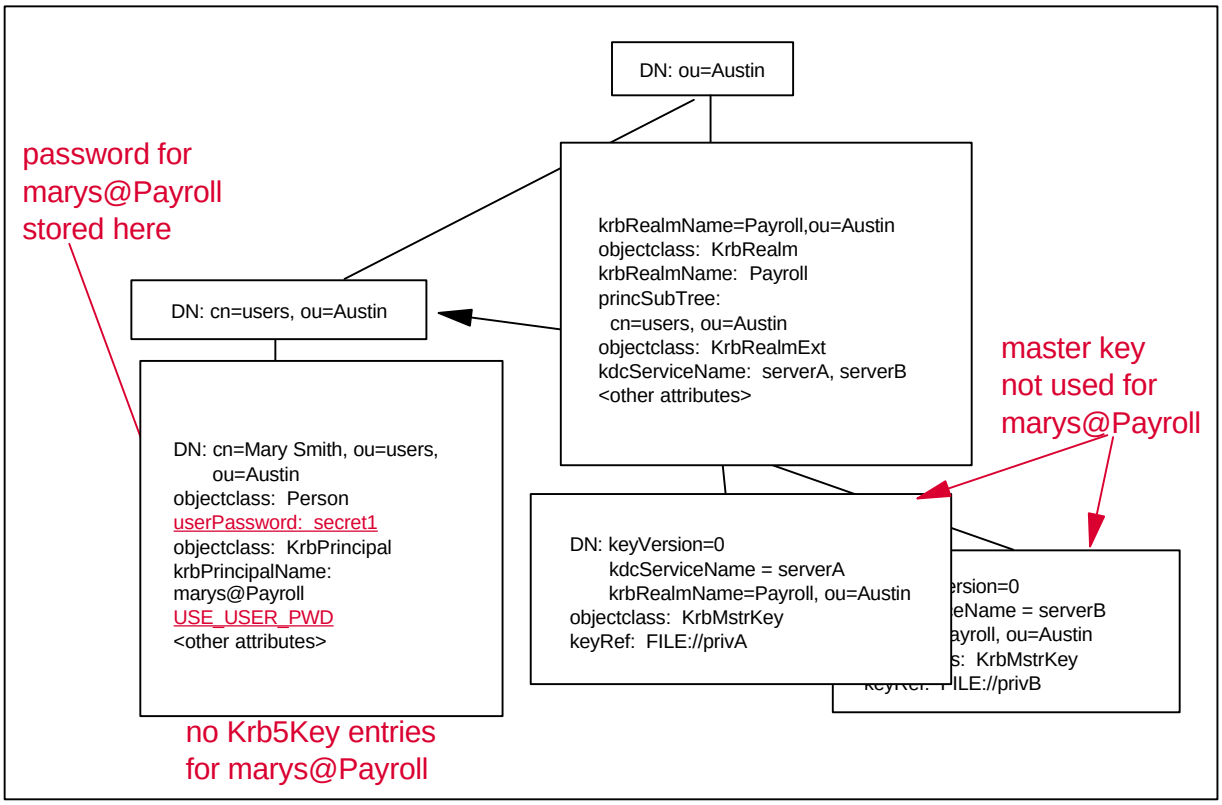
This schema defines an optional USE_USER_PWD flag, which can be configured in the krb-Attributes attribute of a principal or realm entry. This flag is provided only for compatibility with LDAP configurations that use the userPassword attribute.

The USE_USER_PWD flag instructs the KDC server to obtain the password for the principal from the userPassword attribute rather than from a KrbKey entry. This makes it possible for a user to configure a password for a principal entry using LDAP interfaces. It also makes it possible for a user to use the same password with Kerberos that the user is using with other applications supporting the userPassword attribute.

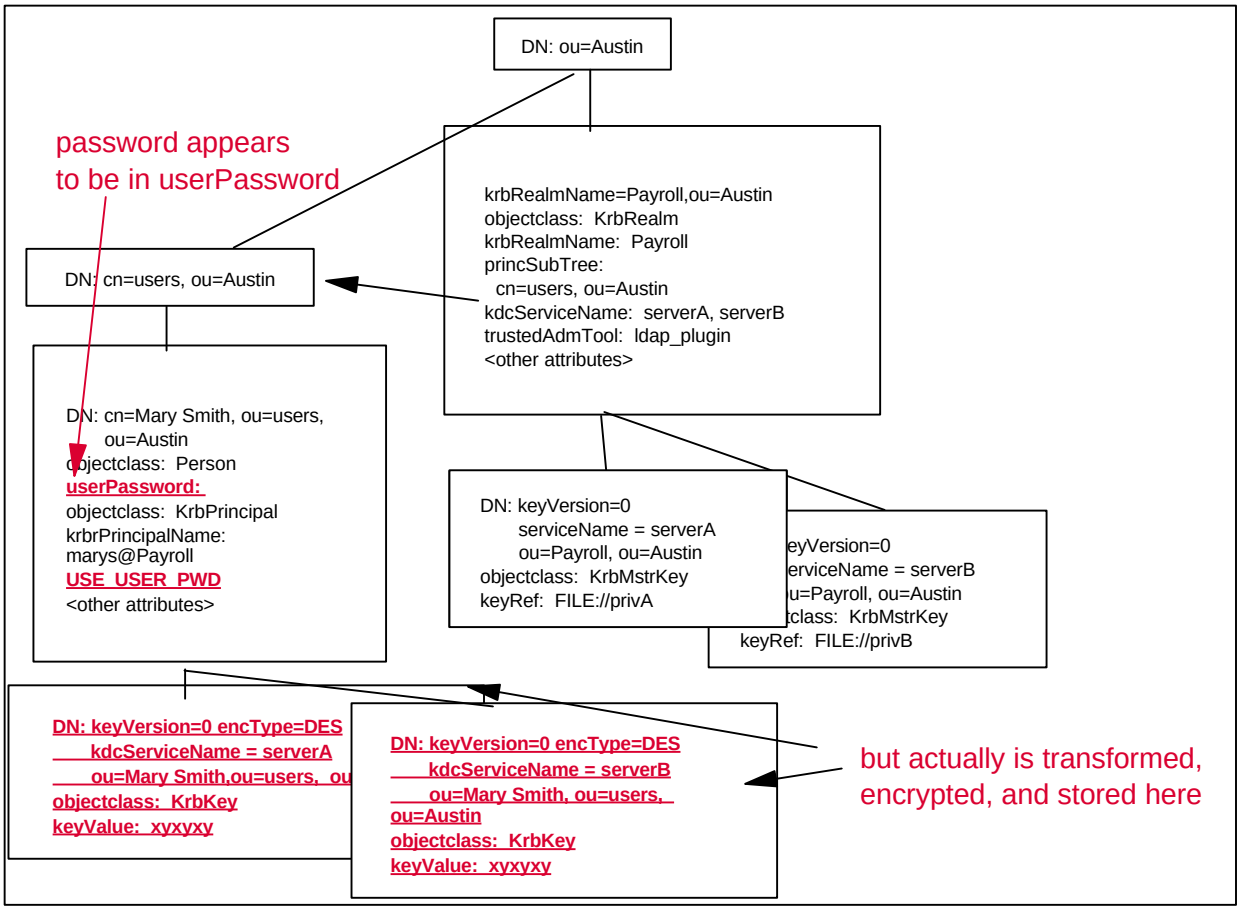
An administrator can configure the USE_USER_PWD flag only if following conditions are met:

- The LDAP server supports retrieving the original plain-text password from the userPassword attribute.
- The LDAP administrator configured the userPassword attribute so that the plain-text password will be retrieved from this attribute.
- The KDC servers in the realm support obtaining the password from the userPassword attribute.
- The administrator is comfortable with the security implications of this configuration.

In the following example, the policy for Mary Smith is configured with the USE_USER_PWD attribute. Therefore, Mary Smith can use the password stored in the userPassword attribute with her Kerberos principal identity.



If the LDAP server supports a plug-in to LDAP calls, it would be possible to develop a plug-in that would intercept LDAP calls to the userPassword attribute. The plug-in would make it appear to LDAP users as if the password was stored in plain-text form in the userPassword attribute. However, in reality, the password would be stored as an encrypted key in one or more KrbKey entries. The following figure illustrates this:


DN: ou=Austin
password appears
to be in userPassword
krbRealmName=Payroll,ou=Austin
objectclass: KrbRealm
krbRealmName: Payroll
princSubTree:
cn=users, ou=Austin
kdcServiceName: serverA, serverB
trustedAdmTool: ldap_plugin
<other attributes>
DN: cn=users, ou=Austin
DN: cn=Mary Smith, ou=users,
ou=Austin
objectclass: Person
userPassword:
objectclass: KrbPrincipal
krbrPrincipalName:
marys@Payroll
USE_USER_PWD
<other attributes>
DN: keyVersion=0
serviceName = serverA
ou=Payroll, ou=Austin
objectclass: KrbMstrKey
keyRef: FILE://privA
eyVersion=0
erviceName = serverB
u=Payroll, ou=Austin
class: KrbMstrKey
keyRef: FILE://privB
DN: keyVersion=0 encType=DES
kdcServiceName = serverA
ou=Mary Smith,ou=users, ou
objectclass: KrbKey
keyValue: xyxyxy
DN: keyVersion=0 encType=DES
kdcServiceName = serverB
ou=Mary Smith, ou=users,
ou=Austin
objectclass: KrbKey
keyValue: xyxyxy
but actually is transformed,
encrypted, and stored here

6. Security Considerations

The following describes how the attributes defined in this schema must be protected.

6.1 ACL Protection

All attributes in this schema must be protected through the use of LDAP ACLs. The following describes how these ACLs need to be configured.

The KrbKey entry can be created only by a Kerberos or trusted administration tool. (The KDC will ignore a KrbKey entry created by any other identity.) When a Kerberos or trusted administration tool creates a KrbKey entry, it needs to set the ACLs on the KrbKey entry so that:

- the attributes in the KrbKey entry can be read only by the KDC.
- the attributes in the KrbKey entry can be modified only by a Kerberos or trusted administration tool.

The KrbLog entry can be created only by a KDC server in the realm. (The KDC will ignore a KrbLog entry created by any other identity.) When the KDC creates a KrbLog entry, it needs to set the ACLs on this entry so the entry can be modified only by a KDC server in the realm.

It is the responsibility of the administrator to set the ACLs on the remaining entries. The following are recommendations as to how to set these ACLs:

- Set the ACLs on the attributes defined in KrbPrincipal, KrbMstrKey, KrbPolicy, KrbRealm, and KrbRealmExt so that only a trusted administrator can modify these attributes. (If the LDAP server does not support ACLs at the attribute level, the administrator needs to configure KrbPrincipal attributes in an alias entry rather than in an existing user entry.)
- If the administrator configures the value of the master key in the keyValue attribute of KrbMstrKey, set the ACL on this attribute so that it can be read only by the KDC. (The administrator can alternately configure the value of the master key in another location, as described next.)

6.2 Data Privacy Protection

The privacy of the master key must be protected. An administrator can do this using one of two methods. The recommended method is to encrypt the master key using Kerberos administration tools and then store it in a private location, such as a private file on the KDC server. The administrator specifies the location of the master key in the keyRef attribute of the KrbMstrKey entry. An alternate method is to encrypt the master key using Kerberos administration tools and then store it on LDAP.

The privacy of each principal's password must be protected. An administrator can do this using one of two methods. The recommended method is to store each password as an encrypted key in a KrbKey entry. An alternate method for less critical principals is to store the password in the LDAP userPassword attribute and rely on the LDAP server to encrypt the value stored in userPassword. If the LDAP userPassword attribute is used, the KDC server will need to retrieve the value stored in userPassword in cleartext format. It is the responsibility of the user to configure the LDAP server to encrypt and decrypt the value in userPassword in such a way that the value can be retrieved by the KDC in cleartext format. The user should ensure that the encryption type is DES or higher.

6.3 Protection During Transmission

If the KDC and/or administration tools will be separate from LDAP, it is the responsibility of the user to configure the security protocol for LDAP bind operations between the KDC and the LDAP server and between the administration tools and the LDAP server. It is recommended that the user choose a security protocol such as SSL that offers client/server authentication technique that is as strong or stronger than Kerberos.

The KDC and the administration tools are responsible for encrypting keys before sending the keys to the LDAP server. The KDC or administration tools need to use an encryption type that is as strong or stronger than DES.

7. Definitions of Attributes and Object Classes

This section provides definitions of attributes and object classes used in this schema..

7.1 Attribute Types

This sub-section provides definitions of all the attribute types used in this schema. This includes:

- New attribute types defined in this schema
- Attribute types defined in the Netscape schema
- Attribute types defined in the Tivoli Policy Director schema
- Attribute types defined in the Microsoft Active Directory schema

7.1.1 New Attribute Types Defined in this Schema

```
(
 admAclDB-oid
 NAME 'admAclDB'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 (DN)
 SINGLE-VALUE
 DESC 'The location of an ACL database for a Kerberos administration service,  The location
must be specified as in URL format; i.e., FILE://path/filename.'
 EQUALITY distinguishedNameMatch
)
```

Note: admDictDB is replaced by passwordMinDiffChars.

```
 (
 admKeyLocation-oid
 NAME 'admKeyLocation'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 (DN)
 SINGLE-VALUE
 DESC 'The location of a keytab file containing the key used by the Kerberos administration
service,  The location must be specified as in URL format; i.e., FILE://path/filename.'
 EQUALITY distinguishedNameMatch
)

 (
 admPortNumber-oid
 NAME 'admPortNumber'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
 DESC A list of 'TCP/IP port numbers for use by Kerberos administration service.'
 )

(
 curKeyVersionArray-oid
 NAME 'curKeyVersionArray'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)

 DESC 'An array of current key versions for each encryption type and salt type.  Each current key
in curKeyVersion corresponds to an encryption type in the encTypeArray and a salt type in the
saltTypeArray.'
)
```

(
encType-oid
NAME 'encType'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'A value defining the encryption type of a secret key. Possible values are:

- ENCTYPE_NULL 0x0000
- ENCTYPE_DES_CBC_CRC 0x0001 /* DES cbc mode with CRC-32 */
- ENCTYPE_DES_CBC_MD4 0x0002 /* DES cbc mode with RSA-MD4 */
- ENCTYPE_DES_CBC_MD5 0x0003 /* DES cbc mode with RSA-MD5 */
- ENCTYPE_DES_CBC_RAW 0x0004 /* DES cbc mode raw */
- ENCTYPE_DES3_CBC_SHA 0x0005 /* DES-3 cbc mode with NIST-SHA */
- ENCTYPE_DES3_CBC_RAW 0x0006 /* DES-3 cbc mode raw */
- ENCTYPE_RSA_PRIVKEY /* RSA private key; required for support of DCE */
- ENCTYPE_UNKNOWN'

)

(
encTypeArray-oid
NAME 'encTypeArray'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
DESC 'An array of encryption type values. See encType for encryption type values. The KDC couples each encryption type specified in the encTypeArray attribute with a salt type specified in the saltTypeArray attribute to derive an encryption type / salt type value. When a principal sets or changes a password, the KDC will generate from the password one or more keys using each configured encryption type / salt type value. For example, if two encryption type / salt type values are configured, the KDC will generate from the password one key using the first encryption type / salt type and a second key using the second encryption type / salt type.'
)

(
encTypeSupport-oid
NAME 'encTypeSupport'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
DESC 'A list of supported encryption type values. See encType for encryption type values.'
)

(
kdcPortNumber-oid
NAME 'kdcPortNumber'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
DESC 'A List of TCP/IP port numbers for KDC service.'
)

(
kdcServiceName-oid
NAME 'kdcServiceName'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 (directory string)
DESC 'A list of KDC server names. '
EQUALITY caseIgnoreMatch
)

(
keyExpires-oid

```
 NAME 'keyExpires'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
 SINGLE-VALUE
 DESC 'A value indicating the date and time when a key expires.''
)

 (
 keyName-oid
 NAME 'keyName'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 (directory string)
 SINGLE-VALUE
 DESC 'Name of a secret key.'
 EQUALITY caseIgnoreMatch
 )

(
 keyRef-oid
 NAME 'keyRef'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 (DN)
 SINGLE-VALUE
 DESC 'Location (specified in URL format) of secret key.'
 EQUALITY distringuishedNameMatch
 )

(
 keyValue-oid
 NAME 'keyValue'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 (IA5 string)
 SINGLE-VALUE
 DESC 'Value of a secret key.'
 EQUALITY caseExactMatch
)

(
 keyVersion-oid
 NAME 'keyVersion'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
 SINGLE-VALUE
 DESC 'Version of a secret key; a monotomic increasing number beginning with 1.'
 )
(
krbAliasedObjectName-oid
 NAME 'krbAliasedObjectName'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 (DN)
 SINGLE-VALUE
 DESC 'Forward reference to the entry for which this entry is an alias.'
 EQUALITY distinghishedNameMatch
)

(
krbAliases-oid
 NAME 'krbAliases'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 (DN)
 DESC 'A list of backward references to entries that can serve as aliases for this entry.'
 EQUALITY distinghishedNameMatch
)
```

(
krb-Attributes-oid
NAME 'krb-Attributes'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC "A value containing one or more attributes that apply to an account. Each attribute is set with a flag. The following flags are available:

- KRB5_KDB_NEW_PRINC
- KRB5_KDB_PWCHANGE_SERVICE
- KRB5_KDB_REQUIRES_HW_AUTH
- KRB5_KDB_REQUIRES_PWCHANGE
- KRB5_KDB_SUPPORT_DESMD5
- KRB5_KDB_DISALLOW_DUB_SKEY
- KRB5_KDB_DISALLOW_POSTDATED
- KRB5_KDB_DIALLOW_PROXIABLE
- KRB5_KDB_DISALLOW_RENEWABLE
- KRB5_KDB_DIALLOW_TGT_BASED
- USER_TO_USER
- KRB5_KDB_DISALLOW_SVR
- USE_USER_PASSWORD'

)

(
krbCreatorsName-oid
NAME 'krbCreatorsName'
SYNTAX (DirectoryString)
SINGLE-VALUE
DESC 'The identity that created the Kerberos principal named in entry. (This entry is the entry containing the krbCreatorsName attribute. The Kerberos principal named in this entry is the principal named in the krbPrincipalName attribute of this entry. The identity that created the Kerberos principal named in this entry is the identity that first added the krbPrincipalName attribute to this entry.) It is the responsibility of the Kerberos administrator or configuration tools to add the krbCreatorsName attribute to this entry and protect this attribute so it cannot be modified by untrusted identities. If this entry does not contain a krbCreatorsName attribute, the LDAP system-controlled creatorsName attribute is assumed to contain the correct creator identity.'
EQUALITY caseIgnoreMatch
)

(
createTimestamp-oid
NAME 'createTimestamp'
SYNTAX (generalizedTime)
DESC 'The date and time when the identity stored in the krbCreatorsName attribute created the Kerberos principal named in this entry. It is the responsibility of the Kerberos administrator or configuration tools to configure the krbCreateTimestamp attribute and to protect this attribute so it cannot be modified. If this entry does not contain a krbCreatorsName attribute, the LDAP system-controlled krbCreateTimestamp attribute is assumed to contain the correct creation date.'
)

(
krbExtraData-oid
NAME 'krbExtraData'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 (IA5 string)
SINGLE-VALUE

DESC 'Extra data that is associated with a Kerberos principal and that has an application-specific meaning. This attribute is provided to support the Kerberos kadmin APIs.'
EQUALITY ‘caseExactMatch’
)

(
krbModifiersName-oid
NAME 'krbModifiersName'
SYNTAX (DirectoryString)
SINGLE-VALUE
DESC 'The last modifier of any attribute associated with the Kerberos principal named in this entry. ("This" entry is the entry containing the krbModifiersName attribute. The Kerberos principal named in this entry is the principal named in the krbPrincipalName attribute of this entry.) It is the responsibility of the Kerberos administrator or configuration tools to update the krbLastModified attribute and to protect this attribute so it cannot be configured by untrusted identities. If this entry does not contain the krbLastModifiersName attribute, the LDAP system-controlled attribute of this entry is used to get the identity that last modified this entry.'
EQUALITY caseIgnoreMatch
)

(
krbModifyTimestamp-oid
NAME 'krbModifyTimestamp'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 (GeneralizedTime)
SINGLE-VALUE
DESC 'The date and time when the identity specified in the krbModifiersName attribute made the last modification. It is the responsibility of the Kerberos administrator or configuration tools to update the krbModifyTimestamp attribute and to protect it so it cannot be configured by untrusted identities. If this entry does not contain a krbModifiersName attribute, the modifyTimestamp attribute is used to get the date of the last modification to this entry.'
)

(
krbPrincipalName-oid
NAME 'krbPrincipalName'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 (directory string)
SINGLE-VALUE
DESC 'Kerberos principal identity for a user in the form <principal>@<realm>. This attribute is defined on Active Directory; OID is 1.2.840.113556.1.4.656'
EQUALITY caseExactMatch
)

(
krbRealmName-oid
NAME 'krbRealmName'
SYNTAX 1.3.6. 11.4.1.1466.115.121.1.15 (directory string)
SINGLE-VALUE
DESC 'Name of a security realm.'
EQUALITY caseExactMatch
)

(
krbTaggedDataList-oid
NAME 'krbTaggedDataList'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.5 (binary)

DESC 'List of tagged data structures that is associated with a Kerberos principal and that is defined by a Kerberos kadmin application. This attribute is provided to support the Kerberos kadmin APIs.'
)

Note: logType is replaced by cn=KrbLog.

Note: minPwdClasses is replaced by passwordMinDiffChars.
(
mstrKeyVersion-oid
NAME 'mstrKeyVersion'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'Version of a master secret key that was used to encrypt a user secret key.'
EQUALITY integerFirstComponentMatch
)

(
multKeyVersionsOK-oid
NAME 'multKeyVersionsOK'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 (boolean)
SINGLE-VALUE
DESC 'True if multiple versions of a key for each encryption type can be stored for this account.'
)

(
nextKeyVersion-oid
NAME 'nextKeyVersion'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'Next version of a secret key.'
)

(
policyObject-oid
NAME 'policyObject'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 (DN)
SINGLE-VALUE
DESC 'Forward reference to an entry containing policy information.'
EQUALITY distinguishedNameMatch
)

(
principalType-oid
NAME 'principalType'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'Value defining the type of an principal. The available principal type values are:

- KRB5_NT_UNKNOWN 0
- KRB5_NT_PRINCIPAL 1
- KRB5_NT_SRV_INST 2
- KRB5_NT_SRV_HST 3
- KRB5_NT_SRV_XHST 4
- KRB5_NT_UID 5'

)

(
princSubTree-oid
NAME 'princSubTree
SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 (DN)
DESC 'A list of forward references to an entry that starts a subtree where principals are configured for this realm.'
EQUALITY distinghishedNameMatch

(
saltType-oid
NAME 'saltType'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'A value indicating the type of an associated salt value. The type indicates how the salt value was determined. The available salt types are:

- ·KRB5_KDB_SALTTYPE_NORMAL : 0
- ·KRB5_KDB_SALTTYPE_V4: 1
- KRB5_KDB_SALTTYPE_NOREALM: 2
- KRB5_KDB_SALTTYPE_ONLYREALM: 3
- KRB5_KDB_SALTTYPE_SPECIAL: 4
- KRB5_KDB_SALTTYPE_AFS3:5'

)

(
saltTypeArray-oid
NAME 'saltTypeArray'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
DESC 'An array of salt type values. See saltType for supported salt type values. The KDC couples each encryption type specified in the encTypeArray attribute with a salt type specified in the saltTypeArray attribute to derive an encryption type / salt type value. When a principal sets or changes a password, the KDC will generate from the password one or more keys using each configured encryption type / salt type value. For example, if two encryption type / salt type values are configured, the KDC will generate from the password one key using the first encryption type / salt type and a second key using the second encryption type / salt type.'
)

(
saltTypeSupport-oid
NAME 'saltTypeSupport'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
DESC 'List of values defining the supported salt types.'
)

(
saltValue-oid
NAME 'saltValue'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 (IA5 string)
SINGLE-VALUE
DESC 'Value of a salt. A salt is used as a seed or offset to the algorithm used to generate a key from a password.'
EQUALITY caseExactMatch
)

(

trustedAdmTool-oid
NAME 'trustedAdmTool'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 (DN)
DESC 'A list of trusted administration tools. '
EQUALITY distinguishedNameMatch
)

7.1.2 Attribute Types Defined in the Netscape Schema

(
passwordDictFiles
)

(
passwordMaxAge
)

(
passwordMinAge
)

(
passwordMinDiffChars
)

(
passwordMinLength
)

7.1.3 Attribute Types Defined in the Microsoft Active Directory Schema

(
1.2.840.113556.1.4.159
accountExpires-oid
NAME 'accountExpires'
SYNTAX 1.2.840.113556.1.4.906 (interval)
SINGLE-VALUE
DESC 'Value used to compute date and time when account will expire. (This attribute is defined in the Microsoft Active Directory schema; OID is 1.2.840.113556.1.4.159.')
)

(
1.2.840.113556.1.4.49
NAME 'badPasswordTime'
SYNTAX 1.2.840.113556.1.4.906 (interval)
SINGLE-VALUE
DESC 'Value used to compute date and time of last unsuccessful logon attempt. This attribute is defined in the Microsoft Active Directory schema; OID is 1.2.840.113556.1.4.49')
)

(
1.2.840.113556.1.4.12

NAME 'badPwdCount'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'Number of unsuccessful attempts to authenticate with this account. This attribute is defined in the Microsoft Active Directory schema; OID is 1.2.840.113556.1.4.12'
)

(
1.2.840.113556.1.4.52
NAME 'lastLogon'
SYNTAX 1.2.840.113556.1.4.906 (interval)
SINGLE-VALUE
DESC 'A value used to compute date and time of last successful logon. This attribute is defined in the Microsoft Active Directory schema. OID is 1.2.840.113556.1.4.52'
)
(
1.2.840.113556.1.4.95
NAME 'pwdHistoryLength'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'Number of previous versions of passwords that are stored. This attribute id defined in the Microsoft Active Directory schema; OID is 1.2.840.113556.1.4.95'
)

(
1.2.840.113556.1.4.96
NAME 'pwdLastSet'
SYNTAX 1.2.840.113556.1.4.906 (interval)
SINGLE-VALUE
DESC 'A value defining the date and time when the password was last set. This attribute is defined in the Microsoft Active Directory schema; OID is 1.2.840.113556.1.4.96'
)

(
1.2.840.113556.1.4.74
NAME 'maxPwdAge'
SYNTAX 1.2.840.113556.1.4.906 (interval)
SINGLE-VALUE
DESC 'A value defining the maximum age of a password. Defined in Active Directory; OID is 1.2.840.113556.1.4.74'
)
(
1.2.840.113556.1.4.75
NAME 'maxRenewAge'
SYNTAX 1.2.840.113556.1.4.906 (interval)
SINGLE-VALUE
DESC 'Value defining the maximum renewable lifetime of a ticket. This attribute is defined in Microsoft Active Directory schema; OID is 1.2.840.113556.1.4.75'
)

(
1.2.840.113556.1.4.77
NAME 'maxTicketAge'
SYNTAX 1.2.840.113556.1.4.906 (interval)
SINGLE-VALUE

DESC 'Value defining the maximum lifetime of a user ticket. This attribute is defined in the Microsoft Active Directory schema; OID is 1.2.840.113556.1.4.77'
)

(
1.2.840.113556.1.4.78
NAME 'minPwdAge'
SYNTAX 1.2.840.113556.1.4.906 (interval)
SINGLE-VALUE
DESC 'Value used to compute minimum lifetime of a password. This attribute is defined in Microsoft Active Directory schema; OID is 1.2.840.113556.1.4.78'
)

(
1.2.840.113556.1.4.79
NAME 'minPwdLength'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'Value defining the minimum length of a password. Defined in Active Directory; OID is 1.2.840.113556.1.4.79'
)

(
1.2.840.113556.1.4.8
NAME 'userAccountControl'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 (integer)
SINGLE-VALUE
DESC 'A value containing one or more attributes that apply to an account. This attribute is defined in the Microsoft Active Directory schema. Each attribute is set with a flag. The following flags, which are defined in the Microsoft lmaccess.h file, are used in the Kerberos KDC Version 5 LDAP schema:

- UF_ACCOUNT_DISABLE = 0x0001
- UF_DONT_EXPIRE_PASSWD = 0x10000
- UF_TRUSTED_FOR_DELEGATION = 0x80000
- UF_USE_DES_KEY_ONLY = 0x200000
- UF_DONT_REQUIRE_PREAUTH = 0x400000'

)

7.1.4 Attribute Types Defined in the Tivoli/IBM Policy Director Schema

```
attributetypes: (
  1.3.6.1.4.1.4228.1.12
  NAME 'secAcctExpires'
  DESC ' '
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 (GeneralizedTime)
  SINGLE-VALUE
 )
```

7.2 Object Classes

objectclasses: (
KrbAlias-oid
NAME 'KrbAlias'

```
 DESC 'Represents a Kerberos principal identity, which can be associated with another entry by
means of the krbAliasedObjectName attribute.  Kerberos ignores the krbAliasedObjectName
attribute.  However, higher level applications can use the krbAliasedObjectName attribute to
associate this Kerberos principal identity with another directory entry.'
 SUP top
 Auxiliary
 MUST (krbAliasedObjectName)
  )

objectclasses: (
 KrbKey-oid
 NAME 'KrbKey'
 DESC 'Represents a secret key associated with a Kerberos principal identity.  Must be created
by the KDC or a trusted Kerberos administration service.'
 SUP top
 Structural
 MUST ( keyVersion $ encType $ kdcServiceName)
 MAY ( keyExpires $ keyValue $ mstrKeyVersion $ nextKeyVersion $ saltValue $ saltType )
 )

objectclasses: (
 KrbLog-oid
 NAME 'KrbLog'
 DESC 'Represents logon activity for an account or a security realm.  Must be created by the
KDC or a trusted Kerberos administration service and the cn attribute must equal "KrbLog."'
 SUP top
 Structural
 MUST ( cn )
 MAY ( badPasswordTime $ badPwdCount $ lastLogon)
 )

objectclasses: (
 KrbMstrKey-oid
 NAME 'KrbMstrKey'
 DESC 'Represents a secret master key owned by a KDC server or a list of KDC servers.'
 SUP KrbKey
 Structural
 MUST (kdcServiceName)
 MAY ( keyName $ keyRef )
 )
objectclasses: (
 KrbPolicy-oid
 NAME 'KrbPolicy'
 DESC 'Auxiliary class used to configure policy attributes for a Kerberos principal identity.'
 SUP top
 Auxiliary
 MAY ( accountExpires $ krb-Attributes $ maxPwdAge $ maxRenewAge $ maxTicketAge $
minPwdAge $ $ minPwdLength $ multKeyVersionsOK $ passwordExpireTime $
passwordDictFiles $ passwordMaxAge $ passwordMinAge $ passwordMinDiffChars $
passwordMinLength $ pwdHistoryLength $ secAcctExpires $ secAcctValid $ userAccountControl)
 )

objectclasses: (
 KrbPrincipal-oid
 NAME 'KrbPrincipal'
 DESC 'Auxiliary class used to configure a Kerberos principal identity.'
```

```
 SUP top
 Auxiliary
 MUST (krbPrincipalName)
 MAY (curKeyVersionArray $ krbCreatorsName $ krbCreateTimestamp $ krbExtraData $
krbAliases $ krbModifiersName $ krbModifyTimestamp $ krbTaggedDataList $ policyObject  $
principalType $ pwdLastSet)
 )

objectclasses: (
 KrbRealm-oid
 NAME 'KrbRealm'
 DESC 'Auxiliary class used to configure a Kerberos realm.'
 SUP top
 Auxiliary
 MUST ( krbRealmName $ princSubTree )
 )

objectclasses: (
 KrbRealmExt-oid
 NAME 'KrbRealmExt'
 DESC 'Auxiliary object used to configure policy attributes for a Kerberos realm.'
 SUP KrbPolicy
 Auxiliary
 MAY ( admAclDB $ admKeyLocation $ admPortNumber $ encTypeArray $ encTypeSupport $
kdcPortNumber $ kdcServiceName $ policyObject $ saltTypeArray $ saltTypeSupport $
trustedAdmTool )
 )
```

8. Mappings

SEE DCE SCHEMA.

9. Examples of Pseudo Code

This section provides examples of pseudo code for the principal portion of the schema.

9.1 Create Principal

Stores a KDC principal record on LDAP.

Inputs:	princ_entry	KDC principal record
	princ_dn	(optional) DN of LDAP entry to represent principal
	policy_dn	(optional) DN of LDAP entry to contain principal policy
	alias_dn	(optional) DN of referenced LDAP entry (used only if this entry is an alias)
	pwd	plaintext password (used only if use_user_pwd is set)
Outputs:	none	

```
int krb5_ldap_create_principal (princ_entry, princ_dn, princ_type, policy_dn, alias_dn)
{

    // get principal and realm names
    get_princ_and_realm_names (princ_entry, princ, realm)

   // determine if princ@realm is unique
   map_princ_to_DN (princ@ realm, dup_princname_dn, realm_dn, subtree_list))
    if (dup_princname_dn)
                return error_dup_princ_name_dn

   // determine if optional flag are set
    use_user_pwd = is_user_pwd_flag_set (princ_entry)

    // if caller did not specify princ_dn, create princ_dn in default location
    // if no alias_dn, use person object; cn=sn=princ; krbPrincipalName=princ
    // if alias_dn, use KrbAlias object; krbAliasedObjectName=alias_dn; krbPrincipalName=princ
    // if (use_use_pwd configure userpassword=pwd
    if (!princ_dn) {    // caller did not specify princ_dn
          if (!subtree_list)   // no subtree list configured {
               add_realm_dn_to_subtree_list
               princ_dn = create_default_dn(princ, realm_dn)          // dn: cn=princ, realm_dn
          } else {                    // subtree list is configured
               princ_dn  = create_default_dn(princ, first_in_subtree_list);
                                                                      // dn: cn=princ; first_subtree
              create_entry_as_and_krb_princ(princ_dn, policy_dn, alias_dn, pwd,
                                                 use_user_pwd, ..);
   }
   // if caller specified princ_dn, make sure it exists and is not yet a principal
   // then add principal attributes to princ_dn
    } else {      // caller supplied princ_dn
          found_person = lookup_person(princ_dn,..., krb_exist_for_this_person)
          if (!found_person)
               return error_no_matching_princ_dn
          else {
               if (krb_princ_exist_for_this_person) {
                     return error_dn_already_a_princ_entry
               create_krb_princ(princ_dn, policy_dn, alias_dn...);
```

```
    }

    // if use_user_password is not set, create KrbKey entry and set ACLs on it
    if (!use_user_pwd)  // this flag is not set (default)
          create_krbkey_under_princ_dn(princ_entry)
           trusted_adm_tool = get_trusted_adm (realm_dn)
           set_acls_on_krbkey    // can be read only by KDC
                                 // can be modified only by kadmin or
                                 // trusted_adm_tool
    }

    // if log information is specified, create KrbLog entry and set ACLs on it
    if (log_info = get_log_info (princ_entry))      // caller supplied krblog info
          create_krblog_under_princ_dn(princ_entry)
          set_acls_on_krblog     // can be modified only by KDC

    // add new subtree to KrbRealm, if necessary
    if (all_is_successful)
           add_new_subtree_as_needed(subtree_list, princ_dn, realm_dn);
}
```

9.2 Get Principal

Gets a KDC principal record from LDAP.

Inputs:	princ_name	name of the principal in the format princ@realm
Outputs:	princ_rec	KDC principal record
	pwd	plaintext password

```
int krb5_ldap_get_princ (princ_name, princ_dn, princ_rec, pwd)
{


    map_princ_to_dn (princ_name, princ_dn, realm_dn, subtree_list)  // get princ_dn
    if (!princ_dn)
           return error_krb_princ_dn_not_found

     // get krb attributes from princ_entry
     // also get password if use_user_pw
     get_krb_attributes_and_pwd_from_princ_dn (princ_dn, princ_rec, pwd)

     // get attributes from krb if it exists, was created by the KDC, and modified
     // only by the KDC
    if (krblog = find (krblog_with_cn=KrbLog_and_KDC_creator_and_modifier))
         get_krb_attributes_from_krblog (princ_dn, princ_rec)

    // get attributes from krbkey if use_user_pwd is not set and
    // if krbkey exists and was created by
    // a Kerberos administration tool or a trusted administration tool
    if (!use_user_pwd ) {
          if (trusted_adm_tool = get_trusted_tools (realm_dn)
                if (krbkey = krbkey_with_correct_creator (trusted_adm_tool)
                     get_krb_attributes_from_krbkey (princ_dn, princ_rec)
    }
```

```
// get policy attributes from policy_dn, if specified
    if (policy_dn)
            get_krb_attributes_from_policy_dn (policy_dn, princ_rec)
}
```

9.3 Delete Principal

Deletes a KDC principal record from LDAP.

Inputs: princ_name (optional) name of the principal in the format princ@realm

Outputs: None

```
int krb5_ldap_delete_principal (princ_name, princ_dn)
{

     map_princ_to_dn (princ_name, princ_dn, realm_dn, subtree_list)  // get princ_dn
     if (!princ_dn)
            return error_krb_princ_dn_not_found

     // delete krblog, krbkey, and princ attributes from princ_dn
    delete_krblog_if_it_exists (princ_dn)
    delete_krbkey_if_it_exists (princ_dn)
    delete_princ_attributes_from_princ_dn (princ_dn)

}
```

9.4 Map Principal Identity to DN

Maps a principal identity ("princ@realm") to a DN.

Input:	princ_name	“princ@realm”
Output:	princ_dn	DN of principal entry
	realm_dn	DN of realm entry
	subtree_list	list of subtrees configured for realm

```
int map_princ_to_dn {princ_name, found_princ_dn) {
    realm_dn = find (realm)
    subtree_list = get_subtree_list (realm_dn)
          for (each subtree in subtree_list)     // loop through each subtree)
                princ_dn = lookup_princ (subtree, princ)
```

9.5 KDC Compare Key with Plaintext Password

KDC routine for comparing a key received from client preauthentication data (in AS_REQ) with a plaintext password retrieved from the LDAP userPassword attribute.

Input:	client_key	client key from AS_REQ
	client_salt	client salt from AS_REQ (if any)
	client_princ_name	client princ@realm name from AS_REQ
	LDAP_pwd	plaintext password from LDAP userPassword
Output:	result	TRUE if the key derived from the plaintext

password equals the client key

```
int compare_key_with_plaintext_password (client_key, client_salt, client_princ_name,
LDAP_pwd, result) {

    // If no salt sent from client, generate salt from princ@realm
     if (!client_salt)
          client_salt = generate_default_salt (client_princ_name)

     // generate compare_key from LDAP password and salt
     krb5_string_to_key (..., &compare_key, LDAP_pwd, client_salt)

     // compare compare_key with client_key
     isEqual (client_key, compare_key)

}
```

9.6 KDC Generate Key from Plaintext Password

KDC routine for generating a key from a plaintext password retrieved from LDAP when USE_USER_PWD is specified. This routine is used to encrypt the AS_REP when a client does not use a preauthentication protocol.

Input:	client_princ_name	client princ@realm name from AS_REQ
	LDAP_pwd	plaintext password from LDAP userPassword
Output:	key	key generated from plaintext password

```
int generate_key_from_plaintext_password {client_princ_name, LDAP_pwd, key) {

    // get default salt from client princ@realm name
    salt = generate_default_salt (client_princ_name)

     // generate key from LDAP password and salt
     krb5_string_to_key (..., &key, LDAP_pwd, salt)
}
```