HP Open Source Security for OpenVMS
Volume 3: Kerberos

Kerberos Version Version 3.0 for OpenVMS, based on MIT Kerberos V5 Release 1.4.1

OpenVMS I64 Version 8.2 or higher
OpenVMS Alpha Version 7.3-2 or higher

This manual supersedes HP Open Source Security for OpenVMS
Volume 3: Kerberos, Version 8.2

Manufacturing Part Number: BA554-90008
July 2006

© Copyright 2006 Hewlett-Packard Development Company, L.P.
Legal Notice

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor’s standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

See Appendix A Open Source Notices for information regarding certain open source code included in this product.

Kerberos is a trademark of the Massachusetts Institute of Technology.

Windows, Windows NT, and MS Windows are U.S. registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group in the U.S. and/or other countries.

All other product names mentioned herein may be trademarks of their respective companies.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

The HP OpenVMS documentation set is available on CD-ROM.

ZK6666
# Contents

1. Introduction to Kerberos
   1.1 Kerberos Terminology ................................................. 37
   1.2 Understanding Kerberos .............................................. 38
     1.2.1 Realms .......................................................... 39
     1.2.2 Security Limitations in Kerberos ......................... 39
   1.3 Kerberos Components ................................................. 40
     1.3.1 KDC ............................................................ 40
     1.3.2 Authentication Service ....................................... 41
     1.3.3 Ticket-Granting Service ................................... 41
     1.3.4 The Kerberos Database ...................................... 41
     1.3.5 Kerberos Utility Programs .................................. 41

2. Installation and Configuration
   2.1 Prerequisites ....................................................... 43
   2.2 Downloading the Kit ............................................... 43
   2.3 Secure Delivery and Kerberos .................................... 44
   2.4 Installing and Configuring Kerberos on OpenVMS Version 8.2 or Higher ..................................................... 44
     2.4.1 Configure HP TCP/IP Services for OpenVMS to Change Hostname Definition to Fully Qualified Domain Name ..................................................... 44
     2.4.2 Configuring Kerberos for OpenVMS on OpenVMS 8.2 or Higher ..................................................... 47
   2.5 Installing and Configuring Kerberos on OpenVMS Alpha Version 7.3-2 ..................................................... 49
   2.6 Configuring Kerberos for OpenVMS Telnet and OpenVMS SSH ..................................................... 51
   2.7 Configuring HP TCP/IP Services for OpenVMS SSH with Kerberos ..................................................... 54
   2.8 Configuring HP TCP/IP Services for OpenVMS Telnet with Kerberos ..................................................... 58
   2.9 Configuring and Starting the Kerberos ACME Agent ..................................................... 62

3. Kerberos Client and Administrative Programs
   3.1 User Client Programs .................................................. 67
     3.1.1 kinit .............................................................. 67
     3.1.2 klist .............................................................. 69
     3.1.3 kdestroy ........................................................ 70
     3.1.4 kpasswd .......................................................... 71
   3.2 Administrative Client Programs .................................... 71
     3.2.1 kadmin and kadmin_local ................................... 71
     3.2.2 kdb5_util ........................................................ 72
     3.2.3 ktutil ............................................................. 75
     3.2.4 kprop ............................................................. 76

4. Kerberos Programming Concepts
   4.1 Overview of Building a Kerberos Application on OpenVMS ..................................................... 81
     4.1.1 Compiling a Kerberos Program on OpenVMS ............... 81
     4.1.2 Linking a Kerberos Program on OpenVMS .................. 81
   4.2 Kerberos Example Programs ....................................... 82
     4.2.1 DCL Example Programs ...................................... 82
     4.2.2 GMAKE Example Programs .................................. 84
5. GSSAPI (Generic Security Services Application Programming Interface)

- gss_accept_sec_context — Establish a security context.......................... 90
  C Prototype .................................................................................. 90
  Arguments .................................................................................... 90
  Description ................................................................................... 92
  Return Values .............................................................................. 93
- gss_acquire_cred — Acquire credential handle .............................. 95
  C Prototype .................................................................................. 95
  Arguments .................................................................................... 95
  Description ................................................................................... 96
  Return Values .............................................................................. 96
- gss_add_cred — Construct credentials incrementally ..................... 97
  C Prototype .................................................................................. 97
  Arguments .................................................................................... 97
  Description ................................................................................... 98
  Return Values .............................................................................. 99
- gss_add_oid_set_member — Add an object identifier to a set ........... 100
  C Prototype .................................................................................. 100
  Arguments .................................................................................... 100
  Description ................................................................................... 100
  Return Values .............................................................................. 100
- gss_compare_name — Allow application to compare two internal names 101
  C Prototype .................................................................................. 101
  Arguments .................................................................................... 101
  Description ................................................................................... 101
  Return Values .............................................................................. 101
- gss_canonicalize_name — Convert internal name to internal mechanism name 102
  C Prototype .................................................................................. 102
  Arguments .................................................................................... 102
  Description ................................................................................... 102
  Return Values .............................................................................. 102
- gss_context_time — Check how much longer context is valid ............. 103
  C Prototype .................................................................................. 103
  Arguments .................................................................................... 103
  Description ................................................................................... 103
  Return Values .............................................................................. 103
- gss_create_empty_oid_set — Create a set containing no object identifiers 104
  C Prototype .................................................................................. 104
  Arguments .................................................................................... 104
  Description ................................................................................... 104
  Return Values .............................................................................. 104
- gss_delete_sec_context — Delete a security context ....................... 105
  C Prototype .................................................................................. 105
  Arguments .................................................................................... 105
  Description ................................................................................... 105
  Return Values .............................................................................. 105
Contents

C Prototype ................................................................. 117
Arguments ................................................................. 117
Description ................................................................. 120
Return Values .............................................................. 121
gss_inquire_context — Extract security context information .................. 123
 C Prototype ................................................................. 123
Arguments ................................................................. 123
Description ................................................................. 125
Return Values .............................................................. 125
gss_inquire_cred — Provide calling application with information about a credential .......................... 126
 C Prototype ................................................................. 126
Arguments ................................................................. 126
Description ................................................................. 126
Return Values .............................................................. 126
gss_inquire_cred_by_mech — Obtain per-mechanism information about a credential ......................... 128
 C Prototype ................................................................. 128
Arguments ................................................................. 128
Description ................................................................. 128
Return Values .............................................................. 129
gss_inquire_names_for_mech — Return set of supported nametypes ...................... 130
 C Prototype ................................................................. 130
Arguments ................................................................. 130
Description ................................................................. 130
Return Values .............................................................. 130
gss_process_context_token — Pass a security context to the security service .................... 131
 C Prototype ................................................................. 131
Arguments ................................................................. 131
Description ................................................................. 131
Return Values .............................................................. 131
gss_release_buffer — Free storage associated with a buffer .................................. 132
 C Prototype ................................................................. 132
Arguments ................................................................. 132
Description ................................................................. 132
Return Values .............................................................. 132
gss_release_cred — Mark a credential for deletion .................................................. 133
 C Prototype ................................................................. 133
Arguments ................................................................. 133
Description ................................................................. 133
Return Values .............................................................. 133
gss_release_name — Free storage associated with an internal name that was allocated by a GSSAPI routine .............................................................. 134
 C Prototype ................................................................. 134
Arguments ................................................................. 134
Description ................................................................. 134
Return Values .............................................................. 134
gss_release_oid_set — Free storage associated with a gss_OID_set object .................. 135
Contents

C Prototype ................................................................. 135
Arguments ................................................................. 135
Description ............................................................... 135
Return Values ............................................................. 135
gss_test_oid_set_member — Determine whether an object identifier is a member of the set .. 136
C Prototype ................................................................. 136
Arguments ................................................................. 136
Description ............................................................... 136
Return Values ............................................................. 136
gss_unwrap — Verify a message with attached MIC and decrypt message content ................. 137
C Prototype ................................................................. 137
Arguments ................................................................. 137
Description ............................................................... 137
Return Values ............................................................. 137
gss_verify_mic — Check that a cryptographic MIC fits the applied message ......................... 139
C Prototype ................................................................. 139
Arguments ................................................................. 139
Description ............................................................... 139
Return Values ............................................................. 139
gss_wrap — Attach a MIC to a message and encrypt the message ........................................... 141
C Prototype ................................................................. 141
Arguments ................................................................. 141
Description ............................................................... 141
Return Values ............................................................. 141
gss_wrap_size_limit — Check expected size of wrapped output ............................................. 143
C Prototype ................................................................. 143
Arguments ................................................................. 143
Description ............................................................... 143
Return Values ............................................................. 143

6. KRB5 (Kerberos V5) Application Programming Interface

krb5_425_conv_principal — Convert a Kerberos V4 principal name to V5 format ................. 146
C Prototype ................................................................. 146
Arguments ................................................................. 146
Description ............................................................... 146
Return Values ............................................................. 146
krb5_524_conv_principal — Separate a Kerberos V5 principal into components .................... 147
C Prototype ................................................................. 147
Arguments ................................................................. 147
Description ............................................................... 147
Return Values ............................................................. 147
krb5_524_convert_creds — Convert Kerberos V5 credentials to V4. ...................................... 148
C Prototype ................................................................. 148
Arguments ................................................................. 148
Description ............................................................... 148
Return Values ............................................................. 148
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_address_compare</td>
<td>Compare two addresses</td>
<td>149</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>krb5_address_order</td>
<td>Return an ordering of two addresses</td>
<td>150</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>150</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>150</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>150</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>150</td>
</tr>
<tr>
<td>krb5_address_search</td>
<td>Search for address in address list</td>
<td>151</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>151</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>151</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>151</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>151</td>
</tr>
<tr>
<td>krb5_aname_to_localname</td>
<td>Convert a principal name to a local name</td>
<td>152</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>krb5_appdefault_boolean</td>
<td>Check Boolean values in appdefault</td>
<td>153</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>krb5_appdefault_string</td>
<td>Check string values in appdefault</td>
<td>154</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>krb5_auth_con_free</td>
<td>Free auth_context</td>
<td>155</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>krb5_auth_con_genaddrs</td>
<td>Get full IP address from address and port</td>
<td>156</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>156</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>156</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>156</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>156</td>
</tr>
<tr>
<td>krb5_auth_con_get_checksum_func</td>
<td>Get the checksum function and data structure</td>
<td>157</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td>157</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td>157</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>157</td>
</tr>
<tr>
<td>Return Values</td>
<td></td>
<td>157</td>
</tr>
<tr>
<td>krb5_auth_con_getrcache</td>
<td>Get the rcache element from the auth_context</td>
<td>158</td>
</tr>
<tr>
<td>C Prototype</td>
<td>Arguments</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>krb5_auth_con_getsendsubkey — Retrieve the send_subkey keyblock from auth_context</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getremoteseqnumber — Retrieve and store the remote sequence number</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getkey — Retrieve keyblock from auth_context</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getauthenticator — Retrieve authenticator used during mutual authentication</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getflags — Retrieve the flags in auth_context</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getlocalseqnumber — Retrieve and store the local sequence number</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getrecvsubkey — Retrieve the recv_subkey keyblock from auth_context</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getaddrs — Retrieve address fields from the auth_context</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>C Prototype</td>
<td>Arguments</td>
<td>Description</td>
</tr>
<tr>
<td>krb5_auth_con_getaddrs</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getrecvsubkey</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getflags</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getlocalseqnumber</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getkey</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getauthenticator</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getaddrs</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getrecvsubkey</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getflags</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getlocalseqnumber</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getkey</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getauthenticator</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getaddrs</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getrecvsubkey</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getflags</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getlocalseqnumber</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getkey</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getauthenticator</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getaddrs</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getrecvsubkey</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getflags</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getlocalseqnumber</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getkey</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getauthenticator</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getaddrs</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getrecvsubkey</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getflags</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_getlocalseqnumber</td>
<td>163</td>
<td></td>
</tr>
</tbody>
</table>
Contents

Arguments .......................................................... 167
Description .......................................................... 167
Return Values ....................................................... 167

krb5_auth_con_set_checksum_func — Set the checksum function and data structure ............................... 168
C Prototype .......................................................... 168
Arguments .......................................................... 168
Description .......................................................... 168
Return Values ....................................................... 168

krb5_auth_con_setaddr — Set address fields in auth_context .......................................................... 169
C Prototype .......................................................... 169
Arguments .......................................................... 169
Description .......................................................... 169
Return Values ....................................................... 169

krb5_auth_con_setflags — Set the flags in auth_context ............................................................... 170
C Prototype .......................................................... 170
Arguments .......................................................... 170
Description .......................................................... 170
Return Values ....................................................... 170

krb5_auth_con_setports — Set port fields in the auth_context ......................................................... 171
C Prototype .......................................................... 171
Arguments .......................................................... 171
Description .......................................................... 171
Return Values ....................................................... 171

krb5_auth_con_setrcache — Set the replay cache .............................................................................. 172
C Prototype .......................................................... 172
Arguments .......................................................... 172
Description .......................................................... 172
Return Values ....................................................... 172

krb5_auth_con_setrecvsubkey — Set the recv_subkey keyblock in auth_context ............................. 173
C Prototype .......................................................... 173
Arguments .......................................................... 173
Description .......................................................... 173
Return Values ....................................................... 173

krb5_auth_con_setsendsubkey — Set the send_subkey keyblock in auth_context ............................ 174
C Prototype .......................................................... 174
Arguments .......................................................... 174
Description .......................................................... 174
Return Values ....................................................... 174

krb5_auth_con_setuseruserkey — Set keyblock field in auth_context to temporary key .................. 175
C Prototype .......................................................... 175
Arguments .......................................................... 175
Description .......................................................... 175
Return Values ....................................................... 175

krb5_build_principal — Build a principal name ................................................................................. 176
C Prototype .......................................................... 176
Arguments .......................................................... 176
### Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_build_principal.va</td>
<td>177</td>
</tr>
<tr>
<td>krb5_c_block_size</td>
<td>178</td>
</tr>
<tr>
<td>krb5_c_checksum_length</td>
<td>179</td>
</tr>
<tr>
<td>krb5_c_decrypt</td>
<td>180</td>
</tr>
<tr>
<td>krb5_c_encrypt</td>
<td>181</td>
</tr>
<tr>
<td>krb5_c_encrypt_length</td>
<td>182</td>
</tr>
<tr>
<td>krb5_c_enctype_compare</td>
<td>183</td>
</tr>
<tr>
<td>krb5_c_is_coll_proof_cksum</td>
<td>184</td>
</tr>
<tr>
<td>krb5_c_is_keyed_cksum</td>
<td>185</td>
</tr>
</tbody>
</table>
Contents

Return Values ................................................................. 185
krb5_c_keyed_checksum_types — Get a list of derived key checksums ......................................................... 186
    C Prototype .......................................................... 186
    Arguments ............................................................ 186
    Description ......................................................... 186
    Return Values ....................................................... 186
krb5_c_make_checksum — Compute a checksum ......................................................................................... 187
    C Prototype .......................................................... 187
    Arguments ............................................................ 187
    Description ......................................................... 187
    Return Values ....................................................... 187
krb5_c_make_random_key — Generate a random key .................................................................................. 188
    C Prototype .......................................................... 188
    Arguments ............................................................ 188
    Description ......................................................... 188
    Return Values ....................................................... 188
krb5_c_random_make_octets — Create random data .................................................................................. 189
    C Prototype .......................................................... 189
    Arguments ............................................................ 189
    Description ......................................................... 189
    Return Values ....................................................... 189
krb5_c_random_seed — Get a random seed ............................................................................................... 190
    C Prototype .......................................................... 190
    Arguments ............................................................ 190
    Description ......................................................... 190
    Return Values ....................................................... 190
krb5_c_string_to_key — Convert a string to a key .................................................................................... 191
    C Prototype .......................................................... 191
    Arguments ............................................................ 191
    Description ......................................................... 191
    Return Values ....................................................... 191
krb5_c_string_to_key_with_params — Convert string key to keyblock ...................................................... 192
    C Prototype .......................................................... 192
    Arguments ............................................................ 192
    Description ......................................................... 192
    Return Values ....................................................... 192
krb5_c_valid_cksumtype — Validate a checksum type ............................................................................... 193
    C Prototype .......................................................... 193
    Arguments ............................................................ 193
    Description ......................................................... 193
    Return Values ....................................................... 193
krb5_c_valid_etype — Validate an encryption type .................................................................................... 194
    C Prototype .......................................................... 194
    Arguments ............................................................ 194
    Description ......................................................... 194
    Return Values ....................................................... 194
## Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_cc_set_flags — Set the flags on the credentials cache</td>
<td>21</td>
</tr>
<tr>
<td>krb5_cc_retrieve_cred — Search the cache for a credential and return it if found</td>
<td>211</td>
</tr>
<tr>
<td>krb5_cc_remove_cred — Remove credentials from the credentials cache</td>
<td>208</td>
</tr>
<tr>
<td>krb5_cc_next_cred — Fetch the next credentials entry</td>
<td>207</td>
</tr>
<tr>
<td>krb5_cc_initialize — Create/refresh a credentials cache</td>
<td>206</td>
</tr>
<tr>
<td>krb5_cc_get_type — Return the CC prefix</td>
<td>205</td>
</tr>
<tr>
<td>krb5_cc_initialize — Create/refresh a credentials cache</td>
<td>206</td>
</tr>
<tr>
<td>krb5_cc_retrieve_cred — Search the cache for a credential and return it if found</td>
<td>211</td>
</tr>
<tr>
<td>krb5_cc_set_default_name — Set default CC name</td>
<td>213</td>
</tr>
<tr>
<td>krb5_cc_set_flags — Set the flags on the credentials cache</td>
<td>214</td>
</tr>
<tr>
<td>krb5_cc_start_seq_get — Start sequential read of cached credentials</td>
<td>215</td>
</tr>
</tbody>
</table>

C Prototype ............................................................................................................. 204
Arguments .................................................................................................................. 204
Description ................................................................................................................ 204
Return Values ............................................................................................................. 204

C Prototype ................................................................................................................ 205
Arguments .................................................................................................................. 205
Description ................................................................................................................ 205
Return Values ............................................................................................................. 205

C Prototype ................................................................................................................ 206
Arguments .................................................................................................................. 206
Description ................................................................................................................ 206
Return Values ............................................................................................................. 206

C Prototype ................................................................................................................ 207
Arguments .................................................................................................................. 207
Description ................................................................................................................ 207
Return Values ............................................................................................................. 207

C Prototype ................................................................................................................ 208
Arguments .................................................................................................................. 208
Description ................................................................................................................ 208
Return Values ............................................................................................................. 209

C Prototype ................................................................................................................ 210
Arguments .................................................................................................................. 210
Description ................................................................................................................ 210
Return Values ............................................................................................................. 210

C Prototype ................................................................................................................ 211
Arguments .................................................................................................................. 211
Description ................................................................................................................ 212
Return Values ............................................................................................................. 212

C Prototype ................................................................................................................ 213
Arguments .................................................................................................................. 213
Description ................................................................................................................ 213
Return Values ............................................................................................................. 213

C Prototype ................................................................................................................ 214
Arguments .................................................................................................................. 214
Description ................................................................................................................ 214
Return Values ............................................................................................................. 214

C Prototype ................................................................................................................ 215
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_copy_creds — Copy a credentials structure</td>
<td>223</td>
</tr>
<tr>
<td>krb5_copy_data — Copy a Kerberos data structure</td>
<td>224</td>
</tr>
<tr>
<td>krb5_copy_checksum — Copy a checksum structure</td>
<td>222</td>
</tr>
<tr>
<td>krb5_copy_authdata — Copy a Kerberos authdata structure</td>
<td>220</td>
</tr>
<tr>
<td>krb5_copy_addresses — Copy Kerberos addresses</td>
<td>219</td>
</tr>
<tr>
<td>krb5_change_password — Change an existing password</td>
<td>217</td>
</tr>
<tr>
<td>krb5_cc_store_cred — Store a credential in the credentials cache</td>
<td>216</td>
</tr>
</tbody>
</table>

**Arguments**

**C Prototype**

**Return Values**

**Description**

**Arguments**

**C Prototype**

**Return Values**

**Description**

**Arguments**

**C Prototype**

**Return Values**

**Description**

**Arguments**

**C Prototype**

**Return Values**

**Description**

**Arguments**

**C Prototype**

**Return Values**

**Description**

**Arguments**

**C Prototype**

**Return Values**

**Description**

**Arguments**

**C Prototype**

**Return Values**

**Description**

**Arguments**

**C Prototype**

**Return Values**

**Description**

**Arguments**

**C Prototype**

**Return Values**

**Description**
## Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_free_ap_rep_enc_part — Free subkey and other data allocated by krb5_rd_rep or krb5_send_auth</td>
<td>233</td>
</tr>
<tr>
<td>krb5_free_addresses — Free a group of addresses.</td>
<td>232</td>
</tr>
<tr>
<td>krb5_enctype_to_string — Convert a Kerberos encryption type value to a string</td>
<td>231</td>
</tr>
<tr>
<td>krb5_deltat_to_string — Convert a Kerberos relative time value to a string</td>
<td>230</td>
</tr>
<tr>
<td>krb5_decode_ticket — Decode a formatted ticket</td>
<td>229</td>
</tr>
<tr>
<td>krb5_copy_ticket — Copy a Kerberos ticket structure.</td>
<td>228</td>
</tr>
<tr>
<td>krb5_copy_principal — Copy a principal structure</td>
<td>227</td>
</tr>
<tr>
<td>krb5_copy_keyblock — Copy a keyblock</td>
<td>226</td>
</tr>
<tr>
<td>krb5_copy_keyblock_contents — Copy a keyblock's contents</td>
<td>226</td>
</tr>
</tbody>
</table>
# Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_free_authdata — Free an authdata structure</td>
<td>234</td>
</tr>
<tr>
<td>krb5_free_authenticator — Free authenticator storage</td>
<td>235</td>
</tr>
<tr>
<td>krb5_free_checksum — Free a checksum</td>
<td>236</td>
</tr>
<tr>
<td>krb5_free_checksum_contents — Free the contents of a checksum structure</td>
<td>237</td>
</tr>
<tr>
<td>krb5_free_cksumtypes — Free a checksum structure</td>
<td>238</td>
</tr>
<tr>
<td>krb5_free_context — Free a context structure</td>
<td>239</td>
</tr>
<tr>
<td>krb5_free_creds — Free credentials</td>
<td>240</td>
</tr>
<tr>
<td>krb5_free_cred_contents — Free credential structures</td>
<td>241</td>
</tr>
<tr>
<td>krb5_free_data — Free storage associated with a krb5_data object</td>
<td>242</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_free_data_contents</td>
<td>Frees contents of a krb5_data structure</td>
<td>243</td>
</tr>
<tr>
<td>krb5_free_default.realm</td>
<td>Free the Kerberos default realm structure</td>
<td>244</td>
</tr>
<tr>
<td>krb5_free_error</td>
<td>Free error information</td>
<td>245</td>
</tr>
<tr>
<td>krb5_free_host.realm</td>
<td>Free storage allocated by krb5_get_host_realm</td>
<td>246</td>
</tr>
<tr>
<td>krb5_free_keyblock</td>
<td>Free keyblock memory</td>
<td>247</td>
</tr>
<tr>
<td>krb5_free_keyblock_contents</td>
<td>Free the contents of a key structure</td>
<td>248</td>
</tr>
<tr>
<td>krb5_free_keytab_entry_contents</td>
<td>Free the contents of a keytab entry</td>
<td>249</td>
</tr>
<tr>
<td>krb5_free_principal</td>
<td>Free the pwd_data allocated by krb5_copy_principal</td>
<td>250</td>
</tr>
<tr>
<td>krb5_free_tgtcreds</td>
<td>Free TGT credentials</td>
<td>251</td>
</tr>
</tbody>
</table>
Contents

- krb5_free_ticket — Free ticket allocated by krb5_copy_ticket
  C Prototype .................................................. 252
  Arguments ...................................................... 252
  Description .................................................... 252
  Return Values ................................................ 252
- krb5_free_unparsed_name — Free a simple name
  C Prototype .................................................. 253
  Arguments ...................................................... 253
  Description .................................................... 253
  Return Values ................................................ 253
- krb5_fwd_tgt_creds — Get a TGT for use at a remote host
  C Prototype .................................................. 254
  Arguments ...................................................... 254
  Description .................................................... 254
  Return Values ................................................ 254
- krb5_get_credentials — Get an additional ticket for the client
  C Prototype .................................................. 255
  Arguments ...................................................... 255
  Description .................................................... 255
  Return Values ................................................ 255
- krb5_get_credentials_renew — Renew a set of existing credentials
  C Prototype .................................................. 257
  Arguments ...................................................... 257
  Description .................................................... 257
  Return Values ................................................ 257
- krb5_get_credentials_validate — Validate a set of existing credentials
  C Prototype .................................................. 258
  Arguments ...................................................... 258
  Description .................................................... 258
  Return Values ................................................ 258
- krb5_get_default_realm — Retrieve the default realm
  C Prototype .................................................. 259
  Arguments ...................................................... 259
  Description .................................................... 259
  Return Values ................................................ 259
- krb5_get_init_creds_keytab — Get initial credentials’ keytab
  C Prototype .................................................. 260
  Arguments ...................................................... 260
  Description .................................................... 260
  Return Values ................................................ 260
- krb5_get_init_creds_opt_init — Initialize options for krb5_get_init_creds* routines
  C Prototype .................................................. 261
  Arguments ...................................................... 261
  Description .................................................... 261
  Return Values ................................................ 261
- krb5_get_init_creds_opt_set_address_list — Set the address list in krb5_get_init_creds_opt
  C Prototype .................................................. 262
  Arguments ...................................................... 262
  Description .................................................... 262
  Return Values ................................................ 262
## Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_init_keyblock — Set up an empty keyblock</td>
<td>281</td>
</tr>
<tr>
<td>krb5_init_context — Initialize a Kerberos context structure</td>
<td>280</td>
</tr>
<tr>
<td>krb5_get_validated_creds — Get validated credentials.</td>
<td>279</td>
</tr>
<tr>
<td>krb5_get_server_rcache — Create a replay cache for server use</td>
<td>277</td>
</tr>
<tr>
<td>krb5_get_renewed_creds — Renew existing credentials</td>
<td>276</td>
</tr>
<tr>
<td>krb5_get_message — Convert an error code into the string representation</td>
<td>273</td>
</tr>
<tr>
<td>krb5_get_permitted_enctypes — Return a list of supported encryption types.</td>
<td>274</td>
</tr>
<tr>
<td>krb5_get_prompt_types — Get prompt_types from the Kerberos context</td>
<td>275</td>
</tr>
</tbody>
</table>

## C Prototype

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_init_keyblock — Set up an empty keyblock</td>
<td>281</td>
</tr>
<tr>
<td>krb5_init_context — Initialize a Kerberos context structure</td>
<td>280</td>
</tr>
<tr>
<td>krb5_get_validated_creds — Get validated credentials.</td>
<td>279</td>
</tr>
<tr>
<td>krb5_get_server_rcache — Create a replay cache for server use</td>
<td>277</td>
</tr>
<tr>
<td>krb5_get_renewed_creds — Renew existing credentials</td>
<td>276</td>
</tr>
<tr>
<td>krb5_get_message — Convert an error code into the string representation</td>
<td>273</td>
</tr>
<tr>
<td>krb5_get_permitted_enctypes — Return a list of supported encryption types.</td>
<td>274</td>
</tr>
<tr>
<td>krb5_get_prompt_types — Get prompt_types from the Kerberos context</td>
<td>275</td>
</tr>
</tbody>
</table>

## Arguments

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_init_keyblock — Set up an empty keyblock</td>
<td>281</td>
</tr>
<tr>
<td>krb5_init_context — Initialize a Kerberos context structure</td>
<td>280</td>
</tr>
<tr>
<td>krb5_get_validated_creds — Get validated credentials.</td>
<td>279</td>
</tr>
<tr>
<td>krb5_get_server_rcache — Create a replay cache for server use</td>
<td>277</td>
</tr>
<tr>
<td>krb5_get_renewed_creds — Renew existing credentials</td>
<td>276</td>
</tr>
<tr>
<td>krb5_get_message — Convert an error code into the string representation</td>
<td>273</td>
</tr>
<tr>
<td>krb5_get_permitted_enctypes — Return a list of supported encryption types.</td>
<td>274</td>
</tr>
<tr>
<td>krb5_get_prompt_types — Get prompt_types from the Kerberos context</td>
<td>275</td>
</tr>
</tbody>
</table>

## Return Values

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_init_keyblock — Set up an empty keyblock</td>
<td>281</td>
</tr>
<tr>
<td>krb5_init_context — Initialize a Kerberos context structure</td>
<td>280</td>
</tr>
<tr>
<td>krb5_get_validated_creds — Get validated credentials.</td>
<td>279</td>
</tr>
<tr>
<td>krb5_get_server_rcache — Create a replay cache for server use</td>
<td>277</td>
</tr>
<tr>
<td>krb5_get_renewed_creds — Renew existing credentials</td>
<td>276</td>
</tr>
<tr>
<td>krb5_get_message — Convert an error code into the string representation</td>
<td>273</td>
</tr>
<tr>
<td>krb5_get_permitted_enctypes — Return a list of supported encryption types.</td>
<td>274</td>
</tr>
<tr>
<td>krb5_get_prompt_types — Get prompt_types from the Kerberos context</td>
<td>275</td>
</tr>
</tbody>
</table>
Contents

Arguments .......................................................... 281
Description ......................................................... 281
Return Values ....................................................... 281

krb5_init_secure_context — Initialize a secure Kerberos context block ................................. 282
C Prototype ......................................................... 282
Arguments ........................................................ 282
Description ......................................................... 282
Return Values ....................................................... 282

krb5_is_thread_safe — Check whether the Kerberos client code supports multithreading ........ 283
C Prototype ......................................................... 283
Description ......................................................... 283
Return Values ....................................................... 283

krb5_kt_add_entry — Add an entry to a key table ................................................................. 284
C Prototype ......................................................... 284
Arguments ........................................................ 284
Description ......................................................... 284
Return Values ....................................................... 284

krb5_kt_close — Close a key table ................................................................. 285
C Prototype ......................................................... 285
Arguments ........................................................ 285
Description ......................................................... 285
Return Values ....................................................... 285

krb5_kt_default — Return a handle to the default keytab ...................................................... 286
C Prototype ......................................................... 286
Arguments ........................................................ 286
Description ......................................................... 286
Return Values ....................................................... 286

krb5_kt_default_name — Get default key table name ............................................................ 287
C Prototype ......................................................... 287
Arguments ........................................................ 287
Description ......................................................... 287
Return Values ....................................................... 287

krb5_kt_end_seq_get — Complete a series of sequential key table entry retrievals ..................... 288
C Prototype ......................................................... 288
Arguments ........................................................ 288
Description ......................................................... 288
Return Values ....................................................... 288

krb5_kt_get_entry — Retrieve an entry from the key table .................................................... 289
C Prototype ......................................................... 289
Arguments ........................................................ 289
Description ......................................................... 289
Return Values ....................................................... 289

krb5_kt_get_name — Get key table name ................................................................. 290
C Prototype ......................................................... 290
Arguments ........................................................ 290
Description ......................................................... 290
Return Values .................................................. 290

krb5_kt_get_type — Return the keytab prefix .................................................. 291
C Prototype .................................................. 291
Arguments .................................................. 291
Description .................................................. 291
Return Values .................................................. 291

krb5_kt_next_entry — Retrieve the next entry from the key table .................. 292
C Prototype .................................................. 292
Arguments .................................................. 292
Description .................................................. 292
Return Values .................................................. 292

krb5_kt_read_service_key — Retrieve a service key from the key table .......... 293
C Prototype .................................................. 293
Arguments .................................................. 293
Description .................................................. 293
Return Values .................................................. 293

krb5_kt_remove_entry — Remove an entry from a key table ......................... 294
C Prototype .................................................. 294
Arguments .................................................. 294
Description .................................................. 294
Return Values .................................................. 294

krb5_kt_resolve — Get keytab handle ................................................................. 295
C Prototype .................................................. 295
Arguments .................................................. 295
Description .................................................. 295
Return Values .................................................. 295

krb5_kt_start_seq_get — Start a sequential retrieve of key table entries .... 296
C Prototype .................................................. 296
Arguments .................................................. 296
Description .................................................. 296
Return Values .................................................. 296

krb5_kuserok — Determine whether the local user is authorized to log in .... 297
C Prototype .................................................. 297
Arguments .................................................. 297
Description .................................................. 297
Return Values .................................................. 297

krb5_mk_1cred — Encode a KRB_CRED message for krb5_rd_cred ............. 298
C Prototype .................................................. 298
Arguments .................................................. 298
Description .................................................. 298
Return Values .................................................. 298

krb5_mk_error — Format an error message ...................................................... 299
C Prototype .................................................. 299
Arguments .................................................. 299
Description .................................................. 299
Return Values .................................................. 299
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Arguments</th>
<th>C Prototype</th>
<th>Return Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_mk_cred</td>
<td>Encode a KRB_CRED message for krb5_rd_cred</td>
<td></td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>C Prototype</td>
<td></td>
<td></td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>krb5_parse_name</td>
<td>Convert string principal name to protocol format</td>
<td></td>
<td></td>
<td>311</td>
</tr>
<tr>
<td>krb5_mk_safe</td>
<td>Format a KRB_SAFE message</td>
<td></td>
<td></td>
<td>308</td>
</tr>
<tr>
<td>krb5_mk_req</td>
<td>Format a KRB_AP_REQ message</td>
<td></td>
<td></td>
<td>304</td>
</tr>
<tr>
<td>krb5_mk_rep</td>
<td>Format and encrypt an AP_REP message</td>
<td></td>
<td></td>
<td>303</td>
</tr>
<tr>
<td>krb5_mk_req_extended</td>
<td>Format a KRB_AP_REQ message with additional options</td>
<td></td>
<td></td>
<td>306</td>
</tr>
<tr>
<td>krb5_mk_safe</td>
<td>Format a KRB_SAFE message</td>
<td></td>
<td></td>
<td>308</td>
</tr>
<tr>
<td>krb5_os_localaddr</td>
<td>Return all protocol addresses of this host</td>
<td></td>
<td></td>
<td>310</td>
</tr>
<tr>
<td>krb5_parse_name</td>
<td>Convert string principal name to protocol format</td>
<td></td>
<td></td>
<td>311</td>
</tr>
<tr>
<td>krb5_principal2salt</td>
<td>Convert a krb5_principal into its default salt</td>
<td></td>
<td></td>
<td>312</td>
</tr>
<tr>
<td>krb5_principal_compare</td>
<td>Compare two principals</td>
<td></td>
<td></td>
<td>313</td>
</tr>
</tbody>
</table>
Contents

C Prototype .......................................................... 313
Arguments .......................................................... 313
Description .......................................................... 313
Return Values ....................................................... 313

krb5_realm_compare — Compare the realms of two principals ........ 324
C Prototype .......................................................... 324
Arguments .......................................................... 324
Description .......................................................... 324
Return Values ....................................................... 324

krb5_read_password — Read a password from the keyboard .......... 323
C Prototype .......................................................... 323
Arguments .......................................................... 323
Description .......................................................... 323
Return Values ....................................................... 323

krb5_rd_safe — Parse a KRB_SAFE message ........................ 321
C Prototype .......................................................... 321
Arguments .......................................................... 321
Description .......................................................... 321
Return Values ....................................................... 321

krb5_rd_req — Parse a KRB_AP_REQ message ........................ 319
C Prototype .......................................................... 319
Arguments .......................................................... 319
Description .......................................................... 319
Return Values ....................................................... 319

krb5_rd_rep — Parse and decrypt an AP_REP message .............. 318
C Prototype .......................................................... 318
Arguments .......................................................... 318
Description .......................................................... 318
Return Values ....................................................... 318

krb5_rd_cred — Read a KRB_CRED message ........................ 315
C Prototype .......................................................... 315
Arguments .......................................................... 315
Description .......................................................... 315
Return Values ....................................................... 315

krb5_rd_error — Read an error protocol message ..................... 316
C Prototype .......................................................... 316
Arguments .......................................................... 316
Description .......................................................... 316
Return Values ....................................................... 316

krb5_rd_priv — Parse a KRB_PRIV message ........................ 317
C Prototype .......................................................... 317
Arguments .......................................................... 317
Description .......................................................... 317
Return Values ....................................................... 317

krb5_rd_rep — Parse and decrypt an AP_REP message .............. 318
C Prototype .......................................................... 318
Arguments .......................................................... 318
Description .......................................................... 318
Return Values ....................................................... 318

krb5_rd_req — Parse a KRB_AP_REQ message ........................ 319
C Prototype .......................................................... 319
Arguments .......................................................... 319
Description .......................................................... 319
Return Values ....................................................... 320

krb5_rd_safe — Parse a KRB_SAFE message ........................ 321
C Prototype .......................................................... 321
Arguments .......................................................... 321
Description .......................................................... 321
Return Values ....................................................... 321

krb5_read_password — Read a password from the keyboard .......... 323
C Prototype .......................................................... 323
Arguments .......................................................... 323
Description .......................................................... 323
Return Values ....................................................... 323

krb5_realm_compare — Compare the realms of two principals ....... 324
C Prototype .......................................................... 324
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_set_principal_realm — Set the realm in the current context.</td>
<td>336</td>
</tr>
<tr>
<td>krb5_set_password_using_ccache — Implements RFC 3244 set password using credentials cache</td>
<td>335</td>
</tr>
<tr>
<td>krb5_set_password — Implements set password per RFC 3244.</td>
<td>334</td>
</tr>
<tr>
<td>krb5_set_default_tgs_enctypes — Set default TGS encryption types.</td>
<td>333</td>
</tr>
<tr>
<td>krb5_recvauth_version — Receive authenticated message with version information.</td>
<td>327</td>
</tr>
<tr>
<td>krb5_recvauth — Receive authenticated message.</td>
<td>325</td>
</tr>
<tr>
<td>krb5_recvauth_version — Receive authenticated message with version information.</td>
<td>327</td>
</tr>
<tr>
<td>krb5_set_default_realm — Sets the default realm</td>
<td>332</td>
</tr>
<tr>
<td>krb5_set_default_tgs_enctypes — Set default TGS encryption types.</td>
<td>333</td>
</tr>
<tr>
<td>krb5_set_password — Implements set password per RFC 3244.</td>
<td>334</td>
</tr>
<tr>
<td>krb5_set_password_using_ccache — Implements RFC 3244 set password using credentials cache</td>
<td>335</td>
</tr>
<tr>
<td>krb5_set_principal_realm — Set the realm in the current context.</td>
<td>336</td>
</tr>
</tbody>
</table>
Contents

Return Values .................................................................................................................. 345
krb5_unparse_name — Convert protocol format principal name to string format ................. 346
  C Prototype .................................................................................................................. 346
  Arguments .................................................................................................................... 346
  Description ................................................................................................................... 346
  Return Values .............................................................................................................. 346
krb5_unparse_name_ext — Convert multiple protocol format principal names to string format .............................................................................................................. 347
  C Prototype .................................................................................................................. 347
  Arguments .................................................................................................................... 347
  Description ................................................................................................................... 347
  Return Values .............................................................................................................. 347
krb5_us_timeofday — Retrieves the system time of day (in seconds and microseconds) ......... 348
  C Prototype .................................................................................................................. 348
  Arguments .................................................................................................................... 348
  Description ................................................................................................................... 348
  Return Values .............................................................................................................. 348
krb5_verify_init_creds — Verify initial credentials .......................................................... 349
  C Prototype .................................................................................................................. 349
  Arguments .................................................................................................................... 349
  Description ................................................................................................................... 349
  Return Values .............................................................................................................. 349
krb5_verify_init_creds_opt_init — Initialize krb5_verify_init_creds_opt structure ............ 350
  C Prototype .................................................................................................................. 350
  Arguments .................................................................................................................... 350
  Description ................................................................................................................... 350
  Return Values .............................................................................................................. 350
krb5_verify_init_creds_opt_set_ap_req_nofail — Initialize the ap_req_nofail field in krb5_verify_init_creds_opt ................................................................. 351
  C Prototype .................................................................................................................. 351
  Arguments .................................................................................................................... 351
  Description ................................................................................................................... 351
  Return Values .............................................................................................................. 351

A. Open Source Notices
  A.1 Acknowledgments ..................................................................................................... 353
  A.2 Kerberos Copyright Notice ..................................................................................... 353
  A.3 OpenVision Technologies Copyright Notice ........................................................ 353
  A.4 University of California Copyright Notice ............................................................. 354

Glossary ............................................................................................................................. 357

Index ................................................................................................................................. 359
Table 6-1. Obsolete and Replacement APIs .......................................................... 145
Tables
Figure 1-1. Interrelationships Among Kerberos Components. ......................... 40
Preface

*HP Open Source Security for OpenVMS, Volume 3: Kerberos* describes how to install, configure, and use Kerberos Version 3.0 for OpenVMS, which is based on MIT Kerberos V5 Release 1.4.1.

The information in this manual applies Kerberos on OpenVMS Industry Standard 64 and OpenVMS Alpha. For information about Kerberos on OpenVMS VAX, see *HP Open Source Security for OpenVMS, Volume 3: Kerberos* for Kerberos Version 2.0 (released with OpenVMS Version 7.3-2).

**Intended Audience**

This document is for application developers who want to implement the Kerberos protocol that uses strong cryptography, so that a client can prove its identity to a server (and vice versa) across an insecure network connection.

**Document Structure**

This manual consists of the following chapters:

Chapter 1 provides an overview of Kerberos.

Chapter 2 contains installation and configuration instructions.

Chapter 3 includes information about client programs.

Chapter 4 is a programming tutorial about how to use Kerberos in your application.

Chapter 5 is a reference section that includes documentation about the GSSAPI.

Chapter 6 is a reference section that includes documentation about the KRB5 APIs.

**Related Documents**

The following HP OpenVMS documents are recommended for further information:

- *HP Open Source Security for OpenVMS, Volume 1: Common Data Security Architecture*
- *HP Open Source Security for OpenVMS, Volume 2: HP SSL for OpenVMS*
- *HP OpenVMS Guide to System Security*

The following MIT Kerberos documents are available from the Kerberos for OpenVMS web site, and in the Kerberos kit in the `KRB$ROOT: [DOCDIR]` directory:

- *Kerberos V5 Application Programming Library* (`LIBRARY.PDF`)
- *Kerberos V5 Implementer’s Guide* (`IMPLEMENT.PDF`)
- *Kerberos V5 Installation Guide* (`INSTALL-GUIDE.PS`)
- *Kerberos V5 System Administrator’s Guide* (`ADMIN-GUIDE.PS`)
- *Kerberos V5 UNIX User’s Guide* (`USER-GUIDE.PS`)
- *Upgrading to Kerberos V5 from Kerberos V4* (`UPG-GUIDE.PS`)

For additional information about OpenVMS products and services, see the following World Wide Web address:

http://www.hp.com/go/openvms/
For information about downloading the latest version of Kerberos for OpenVMS, see the following World Wide Web address:

http://h71000.www7.hp.com/openvms/products/kerberos/

For additional information about Kerberos, see the MIT Kerberos web site at the following World Wide Web address:

http://web.mit.edu/kerberos/www/

Related RFCs

The following RFCs are related to Kerberos and may be of interest.

- RFC 1510 The Kerberos Network Authentication Service
- RFC 1964 The Kerberos Version 5 GSS-API Mechanism
- RFC 2743 GSS-API Version 2, Update 1
- RFC 2744 GSS-API Version 2, C Bindings
- RFC 2942 Telnet Authentication: Kerberos Version 5
- RFC 3129 Requirements for Kerberized Internet Negotiation of Keys
- RFC 3244 Microsoft Windows 2000 Kerberos Change Password and Set Password Protocols
- RFC 3961 Encryption and Checksum Specifications for Kerberos 5
- RFC 3962 Advanced Encryption Standard (AES) Encryption for Kerberos 5
- RFC 4120 The Kerberos Network Authentication Service (V5)

Reader's Comments

HP welcomes your comments on this manual.

Please send comments to either of the following addresses:

   Internet: openvmsdoc@hp.com

Postal Mail:
Hewlett-Packard Company
OSSG Documentation Group
ZKO3-4/U08
110 Spit Brook Road
Nashua, NH 03062-2698

How to Order Additional Documentation

For information about how to order additional documentation, visit the following World Wide Web address:

http://www.hp.com/go/openvms/doc/order/
## Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl/x</td>
<td>A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.</td>
</tr>
<tr>
<td>PF1 x</td>
<td>A sequence such as PF1 x indicates that you must first press and release the key labeled PF1 and then press and release another key (x) or a pointing device button.</td>
</tr>
<tr>
<td>Return</td>
<td>In examples, a key name in bold indicates that you press that key.</td>
</tr>
</tbody>
</table>
| ...        | A horizontal ellipsis in examples indicates one of the following possibilities:  
  – Additional optional arguments in a statement have been omitted.  
  – The preceding item or items can be repeated one or more times.  
  – Additional arguments, values, or other information can be entered. |
<p>| .          | A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed. |
| ()         | In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one. |
| []         | In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement. |
| | In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line. |
| {}         | In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line. |
| <strong>bold type</strong> | Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason. In command or script examples, bold text indicates user input. |
| <em>italic type</em> | Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error <em>number</em>), in command lines (/PRODUCER=<em>name</em>), and in command arguments in text (where <em>(dd)</em> represents the predefined par code for the device type). |
| <strong>UPPERCASE TYPE</strong> | Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege. |</p>
<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX command and pathnames, PC-based commands and folders, and certain elements of the C programming language.</td>
</tr>
<tr>
<td>–</td>
<td>A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.</td>
</tr>
<tr>
<td>numbers</td>
<td>All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.</td>
</tr>
</tbody>
</table>
1 Introduction to Kerberos

Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography. It was developed at the Massachusetts Institute of Technology as part of Project Athena in the mid-1980s. Project Athena’s mandate was to explore diverse uses of computing and to build the knowledge base needed for longer-term strategic decisions about how computers fit into the MIT curriculum.

Kerberos is the name of the three-headed dog that guarded the gates of Hades in Greek mythology. Cerberus, who many argue should be the name used, is the Latin name for the equivalent entity in Roman mythology. Until Kerberos V4, this technology was not available to the general public. Prior versions were for only internal Project Athena use. Kerberos V5, the current implementation, is the first commercial-ready release.

The Kerberos protocol uses strong cryptography, so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and server have used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity.

OpenVMS provides support for both Kerberos clients and servers, beginning with OpenVMS Version 7.3-1. Kerberos Version 3.0 for OpenVMS is based on MIT Kerberos V5 Release 1.4.1.

1.1 Kerberos Terminology

The following are commonly used Kerberos terms and their definitions.

**Key Distribution Center (KDC)**

The Ticket-Granting Service (TGS) and the Authentication Server are usually collectively known as the Key Distribution Center.

**Principal Name**

A principal is a unique identity to which Kerberos can assign tickets. It is analogous to an OpenVMS user. The Kerberos database, which performs a function similar to the UAF file on OpenVMS, stores information about principals.

By convention, a principal name is divided into three parts:

- A primary – For a user, a user name. For a system, the word host.
- The instance – An optional string that qualifies the primary.
- The realm – Generally, the DNS domain name in uppercase letters.

**Realm**

The administrative domain that encompasses Kerberos clients and servers is called a realm. Each Kerberos realm has at least one Kerberos server, zero or more Kerberos slave servers, and any number of clients. The master Kerberos database for that site or administrative domain is stored on the Kerberos server. Slave servers have read-only copies of the database that are periodically propagated from the master server.
Secret vs. Private

Secret and private are often used interchangeably. In this manual, it takes two (or more) to share a secret, therefore a shared DES key is a secret key. A key is private only when no one but its owner knows it. Therefore, in public key cryptosystems, one has a public and a private key.

Tickets

Kerberos tickets, also known as credentials, are a set of electronic information used to verify your identity. Kerberos tickets can be stored in a file, or they may exist only in memory.

The first ticket you obtain is a generic Ticket-Granting Ticket (TGT), which is granted upon your initial login to the Kerberos realm. The TGT allows you to obtain additional tickets that give you permission for specific services.

1.2 Understanding Kerberos

Kerberos performs authentication as a trusted third-party authentication service by using conventional (shared secret key) cryptography. Kerberos provides a means of verifying the identities of principals, without relying on authentication 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.

When you integrate Kerberos into an application, it is important to review how and when Kerberos routines ensure that the application design does not compromise the authentication. For instance, an application is not secure if it uses Kerberos routines only on initiation of a stream-based network connection and assumes the absence of any active attackers who might hijack the stream connection.

The Kerberos protocol code libraries, whose API is described in Chapters 5 and 6, can be used to provide encryption to any application. To add authentication to its transactions, a typical network application adds one or two calls to the Kerberos library, which results in the transmission of the necessary messages to achieve authentication.

The two methods for obtaining credentials—the initial ticket exchange and the TGT exchange—use slightly different protocols and require different API routines. The basic difference an API programmer will see is that the initial request does not require a TGT. It does require the client's secret key, because the reply is sent back encrypted in the client's secret key. Usually this request is for a TGT, and TGT-based exchanges are used from then on. In a TGT exchange, the TGT is sent as part of the request for tickets and the reply is encrypted in the session key from the TGT. For example, once a user's password is used to obtain a TGT, it is not required for subsequent TGT exchanges.

The reply consists of a ticket and a session key, encrypted either in the user's secret key (password) or the TGT session key. The combination of a ticket and a session key is known as a credentials cache. (In Kerberos V4, a credentials cache was called a ticket file.) An application client can use these credentials to authenticate to the application server by sending the ticket and an authenticator to the server. The authenticator is encrypted in the session key of the ticket and contains the name of the client, the name of the server, and the time the authenticator was created.

In order to verify the authentication, the application server decrypts the ticket using its service key, which is known only by the application server and the Kerberos server. Inside the ticket, the Kerberos server had placed the name of the client, the name of the server, a key associated with this ticket, and some additional information. The application server then uses the ticket session key to decrypt the authenticator, and verifies that the information in the authenticator matches the information in the ticket and that the timestamp in the
authenticator is recent (to prevent reply attacks). Because the session key was generated randomly by the Kerberos server and delivered encrypted only in the service key and in a key known only by the user, the application server can be confident that user is really who he or she claims to be, because the user was able to encrypt the authenticator using the correct key.

To provide detection of both replay attacks and message stream modification attacks, the integrity of all the messages exchanged between principals can also be guaranteed by generating and transmitting a collision-proof checksum 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 using the session key.

1.2.1 Realms

The Kerberos protocol operates across organizational boundaries. Each organization that runs 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 use its credentials remotely. 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 ticket-granting ticket for the remote realm's ticket-granting service from its local realm. When that ticket-granting ticket is used, the remote ticket-granting service uses the inter-realm key (which usually differs from its own normal TGS key) to decrypt the ticket-granting ticket and to assure that it 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.

This method can be repeated to authenticate across multiple realms. To build a valid authentication path to a distant realm, the local realm must share an inter-realm key with an intermediate realm that communicates with either the distant realm or yet another intermediate realm.

Realms are typically organized hierarchically. Each realm shares a key with its parent and a different key with each child. If two realms do not directly share an inter-realm key, the hierarchical organization allows an authentication path to be easily constructed. If a hierarchical organization is not used, it may be necessary to consult some database 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. It is important for the end service to know which realms were traversed when deciding how much faith to place in the authentication process. To make this easier, a field in each ticket contains the names of the realms that were involved in authenticating the client.

1.2.2 Security Limitations in Kerberos

When you are designing a secure application, be aware of the following limitations of Kerberos:

- Kerberos does not address denial of service attacks. There are places in the Kerberos protocol 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 normal failure modes for the system) is 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, they can use it to masquerade as that principal or impersonate a server to legitimate principals.

- 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 that are encrypted under a key derived from the user's password.
1.3 Kerberos Components

Figure 1-1 depicts the interrelationship between the various components of Kerberos.

Figure 1-1 Interrelationships Among Kerberos Components

When a client logs in to the realm, an authentication request is sent to the Kerberos Key Distribution Center (KDC). A Ticket-Granting Ticket (TGT) is returned as the result of authentication. When the client application starts, the TGT is used to request an application ticket. The application ticket is then sent to the application server, which verifies the application ticket with the KDC. Normal communication can then begin.

The Kerberos registry can be manipulated in several ways. It is initially created via the `krb$configure` command procedure. Other tools used to access the Kerberos information are:

- `kadmin` – Used for reading or updating the Kerberos registry.
- `kinit` – Creates credentials for a user.
- `klist` – Displays the existing credentials for a user.
- `kdestroy` – Deletes a user's credentials.
- `kpasswd` – Changes a user's Kerberos password.
- `kdb5_util` – Dumps or loads the Kerberos database for save and restore operations.

1.3.1 KDC

Each Kerberos realm will have at least one Kerberos server. This server, the Key Distribution Center, contains the Authentication Service, the Ticket-Granting Service, and the master database for Kerberos. These services are implemented as a single daemon: the KDC (`krb$Krb5Kdc`).
1.3.2 Authentication Service

The authentication service handles user authentication, or the process of verifying that principals are correctly identified. It consists of the security server (or servers) in the KDC (or KDCs), and security clients. A security client communicates with a security server to request information and operations. The security server accesses the registry database to perform queries and updates and to validate user logins.

1.3.3 Ticket-Granting Service

Once authenticated, a principal will be granted a TGT and a ticket session key, which gives the principal the right to use the ticket. This combination of the ticket and its associated key is known as your credentials. A principal's credentials are stored in a credentials cache, which is often just a file in the principal's local directory tree.

1.3.4 The Kerberos Database

The Kerberos database contains all of the realm's Kerberos principals, their passwords, and other administrative information about each principal. Each KDC contains its own copy of the Kerberos database. The master KDC contains the primary copy of the database, which it propagates at regular intervals to the slave KDCs. All database changes are made on the master KDC. Slave KDCs provide ticket-granting services only, with no database administration. This allows clients to continue to obtain tickets when the master KDC is unavailable.

1.3.5 Kerberos Utility Programs

OpenVMS provides three different versions of each of the Kerberos user interface programs: the original UNIX® style, a DCL version, and an X Windows version. The DCL interface for the user utilities (kinit, klist, kdestroy, kpasswd) is invoked by the DCL command:

$ KERBEROS

The DCL interface for the administrative utility (kadmin) is invoked by the DCL command:

$ KERBEROS/ADMIN

Either DCL interface can be modified with an /INTERFACE qualifier to invoke the X Windows version. For example, the command line for the administrative program is as follows:

$ KERBEROS/ADMIN/INTERFACE=DECWINDOWS

DCL help is available within each of the DCL interfaces.

kadmin

The kadmin program allows for the maintenance of Kerberos principals, policies, and service key tables (keytabs).

kinit

The kinit program explicitly obtains Kerberos tickets. Similarly, if a user's Kerberos ticket expires, kinit is used to obtain a new one.
Introduction to Kerberos

Kerberos Components

**klist**

The `klist` program displays the existing tickets for a principal and various details about those tickets, including expiration time.

**kdestroy**

The `kdestroy` program removes all of the tickets for a principal. Because Kerberos tickets can be stolen and because someone who steals a ticket can masquerade as another principal, Kerberos tickets should be destroyed when you are away from your computer.

**kpasswd**

The `kpasswd` program changes a Kerberos principal's password. Passwords should be changed periodically.

**kdb5_util**

The `kdb5_util` program creates, destroys, dumps, and loads the Kerberos database. It also allows the creation of a key stash file, which allows a KDC to authenticate itself to the database utilities. Unlike the Kerberos utility programs (with the exception of `kadmin`), access to `kdb5_util` is generally limited to Kerberos administrators.

**kprop**

The `kprop` program propagates the master KDC database to slave KDC servers.

**ktutil**

The `ktutil` command invokes a menu from which an administrator can read, write, or edit entries in a Kerberos V5 keytab or V4 srvtab file.
2 Installation and Configuration

This chapter contains information about installing and configuring Kerberos for OpenVMS.

NOTE For the latest release notes for the current version of Kerberos for OpenVMS, see the Kerberos for OpenVMS web site at:

http://h71000.www7.hp.com/openvms/products/kerberos/

2.1 Prerequisites

Operating System

- HP OpenVMS Industry Standard 64 Version 8.2 or higher, or
- HP OpenVMS Alpha Version 7.3-2 or higher

TCP/IP Transport

- HP TCP/IP Services for OpenVMS Version 5.6 or higher
  (for Kerberos on OpenVMS I64 and OpenVMS Alpha Version 8.3)
- HP TCP/IP Services for OpenVMS Version 5.5 or higher
  (for Kerberos on OpenVMS I64 and OpenVMS Alpha Version 8.2)
- HP TCP/IP Services for OpenVMS Version 5.4 or higher
  (for Kerberos on OpenVMS Alpha Version 7.3-2)

NOTE If you are running a third-party TCP/IP network product such as MultiNet or TCPware from Process Software Corporation, contact your provider regarding running Kerberos Version 3.0 with their TCP/IP network product.

2.2 Downloading the Kit

Kerberos Version 3.0 is included in the OpenVMS Version 8.3 operating system distribution media. If you are running OpenVMS Version 7.3-2, 8.2, or 8.2-1, you can download and install Kerberos Version 3.0.

To download the Kerberos kit from the OpenVMS web site, fill out and submit the Kerberos for OpenVMS registration form at the following URL:

http://h71000.www7.hp.com/openvms/products/kerberos/
2.3 Secure Delivery and Kerberos

The Kerberos for OpenVMS kit is a self-extracting executable file containing a compressed .PCSI file and an associated encrypted, signed manifest (*.ESW) file.

If you copy the Kerberos kit to another location, keep the Kerberos kit and manifest file in the same directory.

NOTE If you are installing Kerberos Version 3.0 on a version of OpenVMS earlier than Version 8.3, the manifest is ignored.

2.4 Installing and Configuring Kerberos on OpenVMS Version 8.2 or Higher

Kerberos Version 3.0 is automatically installed during the installation of OpenVMS Version 8.3, or during an upgrade from a previous version of OpenVMS to Version 8.3.

2.4.1 Configure HP TCP/IP Services for OpenVMS to Change Hostname Definition to Fully Qualified Domain Name

Before configuring or starting Kerberos, check the HP TCP/IP Services for OpenVMS Local Host Database to determine whether your hostname definition is the short name (for example, node1) or the Fully Qualified Domain Name (FQDN) (for example, node1.hp.com).

NOTE If your hostname definition is the short name, you must run TCPIP$CONFIG to change the definition to the fully qualified name. If your hostname definition is the FQDN, continue to Section 2.4.2.

Example 2-1 contains a log of such a change.

Example 2-1 Changing Hostname Definition from Short Name to Fully Qualified Domain Name

$ TCPIP SHOW HOST/LOCAL NODE1

LOCAL database

Host address Host name
1.2.3.4 node1

$ @SYS$STARTUP:TCPIP$CONFIG

TCP/IP Network Configuration Procedure

This procedure helps you define the parameters required to run HP TCP/IP Services for OpenVMS on this system.

Checking TCP/IP Services for OpenVMS configuration database files.
HP TCP/IP Services for OpenVMS Configuration Menu

Configuration options:

1 - Core environment
2 - Client components
3 - Server components
4 - Optional components
5 - Shutdown HP TCP/IP Services for OpenVMS
6 - Startup HP TCP/IP Services for OpenVMS
7 - Run tests

A - Configure options 1 - 4
[E] - Exit configuration procedure

Enter configuration option: 1

HP TCP/IP Services for OpenVMS Core Environment Configuration Menu

Configuration options:

1 - Domain
2 - Interfaces
3 - Routing
4 - BIND Resolver
5 - Time Zone

A - Configure options 1 - 5
[E] - Exit menu

Enter configuration option: 2

HP TCP/IP Services for OpenVMS Interface & Address Configuration Menu

Hostname Details: Configured=node1, Active=node1

Configuration options:

1 - WE0 Menu (EWA0: TwistedPair 1000mbps)
2 - 1.2.3.4/21 node1 Configured,Active

3 - IE0 Menu (EIA0: TwistedPair 100mbps)
I - Information about your configuration

[E] - Exit menu

Enter configuration option: 2

HP TCP/IP Services for OpenVMS Address Configuration Menu

WE0 1.2.3.4/21 node1 Configured,Active WE0

Configuration options:

1 - Change address
2 - Set "node1" as the default hostname
3 - Delete from configuration database
4 - Remove from live system
5 - Add standby aliases to configuration database (for failSAFE IP)

[E] - Exit menu
Enter configuration option: 1

IPv4 Address may be entered with CIDR bits suffix. 
E.g. For a 16-bit netmask enter 10.0.0.1/16

Enter IPv4 Address [1.2.3.4/21]:
Enter hostname [node1]: node1.hp.com

Requested configuration:
Address : 1.2.3.4/21
Netmask : 255.255.248.0 (CIDR bits: 21)
Hostname : node1.hp.com

* Is this correct [YES]:

"node1" is currently associated with address "1.2.3.4". 
Continuing will associate "node1.hp.com" with "1.2.3.4".

* Continue [NO]: YES
Deleted host node1 from host database
Added hostname node1.hp.com (1.2.3.4) to host database
* Update the address in the configuration database [NO]: YES
Updated address WE0:1.2.3.4 in configuration database
* Update the active address [NO]: YES
WE0: delete active inet address node1.hp.com
Updated active address to be WE0:1.2.3.4

HP TCP/IP Services for OpenVMS Interface & Address Configuration Menu
Hostname Details: Configured=node1, Active=node1

Configuration options:
1 - WE0 Menu (EWA0: TwistedPair 1000mbps)
2 - 1.2.3.4/21 node1.hp.com Configured,Active
3 - IE0 Menu (EIA0: TwistedPair 100mbps)
I - Information about your configuration
[E] - Exit menu

Enter configuration option: E

HP TCP/IP Services for OpenVMS Core Environment Configuration Menu

Configuration options:
1 - Domain
2 - Interfaces
3 - Routing
4 - BIND Resolver
5 - Time Zone
A - Configure options 1 - 5
[E] - Exit menu

Enter configuration option: E

HP TCP/IP Services for OpenVMS Configuration Menu

Configuration options:
2.4.2 Configuring Kerberos for OpenVMS on OpenVMS 8.2 or Higher

If you have not previously configured an earlier version of Kerberos on your system, you must run the configuration program before starting Kerberos.

NOTE If you are reconfiguring Kerberos on a system on which Kerberos was previously configured, you must enter the \texttt{kdestroy} command before you run the configuration command procedure \texttt{SYS$STARTUP:KRB$CONFIGURE.COM}. The \texttt{kdestroy} command is defined in \texttt{KRB$SYMBOLS.COM}.

After you have a valid configuration, start Kerberos with the following command:

\texttt{@SYS$STARTUP:KRB$STARTUP.COM}

Example 2-2 shows a configuration log.

Example 2-2 Kerberos Configuration Log on OpenVMS

\texttt{@SYS$STARTUP:KRB$CONFIGURE}

Kerberos V3.0 for OpenVMS Configuration Menu

Configuration options:

\begin{itemize}
  \item [1] Setup Client configuration
  \item [2] Edit Client configuration
  \item [3] Setup Server configuration
  \item [4] Edit Server configuration
  \item [5] Shutdown Servers
  \item [6] Startup Servers
  \item [E] Exit configuration procedure
\end{itemize}

Enter Option: \texttt{1}

Where will the OpenVMS Kerberos 5 KDC be running [ system ]:
What is the OpenVMS Kerberos 5 default domain [ abc.xyz.com ]:
What is the OpenVMS Kerberos 5 Realm name [SYSTEM.ABC.XYZ.COM]:

Press Return to continue ...

Kerberos V3.0 for OpenVMS Configuration Menu

Configuration options:

1 - Setup Client configuration
2 - Edit Client configuration
3 - Setup Server configuration
4 - Edit Server configuration
5 - Shutdown Servers
6 - Startup Servers
E - Exit configuration procedure

Enter Option: 3

Where will the OpenVMS Kerberos 5 KDC be running [system]:

What is the OpenVMS Kerberos 5 default domain [abc.xyz.com]:

What is the OpenVMS Kerberos 5 Realm name [SYSTEM.ABC.XYZ.COM]:

The type of roles the KDC can perform are:

NO_KDC -- where the KDC will not be run
SINGLE_KDC -- where the KDC is the only one in the realm
MASTER_KDC -- where the KDC is the master of 1 or more other KDCs
SLAVE_KDC -- where the KDC is slave to another KDC

What will be the KDC's role on this node [SINGLE_KDC]:

Create the OpenVMS Kerberos 5 database [Y]:

Creating OpenVMS Kerberos 5 database ...

Initializing database 'krb$root:[krb5kdc]principal' for realm 'SYSTEM.ABC.XYZ.COM',
master key name 'K/M@SYSTEM.ABC.XYZ.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.

Enter KDC database master key:
Re-enter KDC database master key to verify:
Priority: info
No dictionary file specified, continuing without one.

Please enter a default OpenVMS Kerberos 5 administrator [SYSTEM]:

Authenticating as principal SYSTEM/admin@SYSTEM.ABC.XYZ.COM with password.

Enter password for principal "SYSTEM/admin@SYSTEM.ABC.XYZ.COM":
Re-enter password for principal "SYSTEM/admin@SYSTEM.ABC.XYZ.COM":
Principal "SYSTEM/admin@SYSTEM.ABC.XYZ.COM" created.
Priority: info
No dictionary file specified, continuing without one.
WARNING: no policy specified for SYSTEM/admin@SYSTEM.ABC.XYZ.COM; defaulting to no policy
Create OpenVMS Kerberos 5 principals [Y]: N
Authenticating as principal SYSTEM/admin@SYSTEM.ABC.XYZ.COM with password.
Priority: info
No dictionary file specified, continuing without one.

KADMIN_LOCAL: Entry for principal kadmin/admin with kvno 3, encryption type Triple DES cbc mode with HMAC/sha1 added to keytab WRFILE=KRBS$ROOT:[KRBSKDC]KADM5.KEYTAB.

KADMIN_LOCAL: Entry for principal kadmin/admin with kvno 3, encryption type DES cbc mode with CRC-32 added to keytab WRFILE=KRBS$ROOT:[KRBSKDC]KADM5.KEYTAB.

Authenticating as principal SYSTEM/admin@SYSTEM.ABC.XYZ.COM with password.
2.5 Installing and Configuring Kerberos on OpenVMS Alpha Version 7.3-2

Kerberos Version 2.0 was automatically installed during the installation of OpenVMS Version 7.3-2, or during an upgrade from a previous version of OpenVMS to Version 7.3-2. Perform the following steps to upgrade Kerberos to Version 3.0 on OpenVMS Version 7.3-2.
See Example 2-2 for a sample Kerberos configuration log. (The configuration log is the same on OpenVMS I64, OpenVMS Alpha, and OpenVMS VAX.) Example 2-3 shows an installation log on OpenVMS Alpha Version 7.3-2.

1. Download the Kerberos kit from the Kerberos for OpenVMS website at
   http://h71000.www7.hp.com/openvms/products/kerberos/

2. Shut down Kerberos by executing SYS$STARTUP:KRB$SHUTDOWN.COM.

3. Install the Kerberos Version 3.0 kit by entering PRODUCT INSTALL KERBEROS. This command will automatically remove the previously installed version. (You do not need to expand the PCSI$COMPRESSED file; PCSI installs from the compressed kit directly.)

4. Add @SYS$STARTUP:KRB$SYMBOLS to SYS$MANAGER:SYLOGIN.COM, if Kerberos was not previously installed and configured.

5. Execute KRB$CONFIGURE.COM, if Kerberos was not previously installed and configured.

6. Start Kerberos by executing SYS$STARTUP:KRB$STARTUP.COM.

Example 2-3 Kerberos Installation Log on OpenVMS Version 7.3-2

Username: system
Password:

   Last interactive login on Tuesday, November 2, 2005 11:12 AM
   Last non-interactive login on Wednesday, November 3, 2005 02:30 PM

$ @SYS$STARTUP:KRB$SHUTDOWN
$ PRODUCT INSTALL KERBEROS

The following product has been selected:

   HP AXPVMS KERBEROS V3.0 Layered Product

Do you want to continue? [YES]

Configuration phase starting ... 

You will be asked to choose options, if any, for each selected product and for any products that may be installed to satisfy software dependency requirements.

   HP AXPVMS KERBEROS V3.0

Do you want the defaults for all options? [YES]

Do you want to review the options? [NO]

Execution phase starting ...

The following product will be installed to destination:
   HP AXPVMS KERBEROS V3.0 DKA0:[VMS$COMMON.]

The following product will be removed from destination:
   HP AXPVMS KERBEROS V2.1 DKA0:[VMS$COMMON.]

Portion done: 0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%

The following product has been installed:
   HP AXPVMS KERBEROS V3.0 Layered Product

The following product has been removed:
   HP AXPVMS KERBEROS V2.1 Layered Product
If Kerberos will be run on this system, but has not been used previously, you need to perform the following steps.

- Run the Kerberos configuration procedure:
  ```
  @SYS$STARTUP:KRB$CONFIGURE.COM
  ```
- Add the following line to SYS$MANAGER:SYSTARTUP_VMS.COM:
  ```
  $ @SYS$STARTUP:KRB$STARTUP
  ```
- Add the following line to SYS$MANAGER:SYLOGIN.COM:
  ```
  $ @SYS$MANAGER:KRB$SYMBOLS
  ```

## 2.6 Configuring Kerberos for OpenVMS Telnet and OpenVMS SSH

Using Kerberos with TCP/IP SSH for OpenVMS or TCP/IP Telnet for OpenVMS, you can authenticate your SSH or Telnet connections between OpenVMS systems.

An OpenVMS account and a corresponding Kerberos principal are required to use both “Kerberized” Telnet and SSH. For each OpenVMS user you create, create a Kerberos principal that exactly matches (including case) its OpenVMS account name. Passwords do not need to match.

To configure Kerberos to use TCP/IP SSH for OpenVMS or TCP/IP Telnet for OpenVMS, or both, perform the following steps. Then see Section 2.7 or Section 2.8 and follow the instructions in the section that applies to you.

1. **Create the principal.** For the Kerberos configuration, you can use either DCL or UNIX-style commands to create the principal.

   The first example below shows the DCL commands. The second example shows the UNIX-style commands. Both styles of commands are entered on an OpenVMS system.

   **DCL:**
   ```
   $ KERBEROS/ADMIN
   KerberosAdmin> login "SYSTEM/admin"
   Enter password:
   Authenticating as principal SYSTEM/admin with password.
   KerberosAdmin> list principal
   K/M@NODE1.HP.COM
   SYSTEM/admin@NODE1.HP.COM
   kadmind/admin@NODE1.HP.COM
   kadmind/changepw@NODE1.HP.COM
   kadmind/node1@NODE1.HP.COM
   kadmind/history@NODE1.HP.COM
   krbtgt/NODE1.HP.COM@NODE1.HP.COM
   KerberosAdmin> create principal “USER1”
   Authenticating as principal SYSTEM/admin with password.
   WARNING: no policy specified for USER1@NODE1.HP.COM; defaulting to
no policy
Enter password for principal "USER1@NODE1.HP.COM";
Re-enter password for principal "USER1@NODE1.HP.COM";
Principal "USER1@NODE1.HP.COM" created.
KerberosAdmin> list principal
Authenticating as principal SYSTEM/admin with password.
K/M@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
USER1@NODE1.HP.COM
kadmin/admin@NODE1.HP.COM
kadmin/changepw@NODE1.HP.COM
kadmin/node1@NODE1.HP.COM
kadmin/history@NODE1.HP.COM
krbtgt/NODE1.HP.COM@NODE1.HP.COM

UNIX:

$ kinit "SYSTEM/admin"
Password for SYSTEM/admin@NODE1.HP.COM:
$ kadmin
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
Enter password:
KADMIN: listprincs
K/M@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
kadmin/admin@NODE1.HP.COM
kadmin/changepw@NODE1.HP.COM
kadmin/node1@NODE1.HP.COM
kadmin/history@NODE1.HP.COM
krbtgt/NODE1.HP.COM@NODE1.HP.COM
KADMIN: addprinc "USER1"
WARNING: no policy specified for USER1@NODE1.HP.COM; defaulting to no policy
Enter password for principal "USER1@NODE1.HP.COM";
Re-enter password for principal "USER1@NODE1.HP.COM";
Principal "USER1@NODE1.HP.COM" created.
KADMIN: listprincs
K/M@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
USER1@NODE1.HP.COM
kadmin/admin@NODE1.HP.COM
kadmin/changepw@NODE1.HP.COM
kadmin/node1@NODE1.HP.COM
kadmin/history@NODE1.HP.COM
krbtgt/NODE1.HP.COM@NODE1.HP.COM

2. Create the Kerberos host principals. For the Kerberos configuration, you can use either DCL or UNIX-style commands to create the principal. The first example below shows the DCL commands. The second example shows the UNIX-style commands.

DCL:

KerberosAdmin> create principal/random "host/node1.hp.com@NODE1.HP.COM"
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
Principal "host/node1.hp.com@NODE1.HP.COM" created.
KerberosAdmin> create principal/random "host/node1@NODE1.HP.COM"
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
Principal "host/node1@NODE1.HP.COM" created.
KerberosAdmin> list principal
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
Installation and Configuration

Configuring Kerberos for OpenVMS Telnet and OpenVMS SSH

K/M@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
USER1@NODE1.HP.COM
host/node1.hp.com@NODE1.HP.COM
host/node1@NODE1.HP.COM
kadmin/admin@NODE1.HP.COM
kadmin/changepw@NODE1.HP.COM
kadmin/history@NODE1.HP.COM
krbtgt/NODE1.HP.COM@NODE1.HP.COM

KerberosAdmin> create keytab "host/node1.hp.com@NODE1.HP.COM"
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
KRB$KERBEROS: Entry for principal host/node1.hp.com@NODE1.HP.COM with kvno 3, encryption type Triple DES cbc mode with HMAC/sha1 added to keytab WRFILE=krb$root:[etc]krb5.keytab.

KRB$KERBEROS: Entry for principal host/node1.hp.com@NODE1.HP.COM with kvno 3, encryption type DES-CBC-CRC mode with CRC-32 added to keytab WRFILE=krb$root:[etc]krb5.keytab.

KerberosAdmin> create keytab "host/node1@NODE1.HP.COM"
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
KRB$KERBEROS: Entry for principal host/node1@NODE1.HP.COM with kvno 3, encryption type Triple DES cbc mode with HMAC/sha1 added to keytab WRFILE=krb$root:[etc]krb5.keytab.

KRB$KERBEROS: Entry for principal host/node1@NODE1.HP.COM with kvno 3, encryption type DES-CBC-CRC mode with CRC-32 added to keytab WRFILE=krb$root:[etc]krb5.keytab.

KerberosAdmin> list keytab
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
host/node1.hp.com@NODE1.HP.COM (kvno: 3, etype: Triple DES cbc mode with HMAC/sha1)
host/node1@NODE1.HP.COM (kvno: 3, etype: DES cbc mode with CRC-32)
host/node1@NODE1.HP.COM (kvno: 3, etype: Triple DES cbc mode with HMAC/sha1)
host/node1@NODE1.HP.COM (kvno: 3, etype: DES cbc mode with CRC-32)

KerberosAdmin> exit
$

UNIX:

KADMIN: addprinc -randkey "host/node1.hp.com@NODE1.HP.COM"
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
Principal "host/node1.hp.com@NODE1.HP.COM" created.
KADMIN: addprinc -randkey "host/node1@NODE1.HP.COM"
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
Principal "host/node1@NODE1.HP.COM" created.
KADMIN: listprincs
K/M@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
USER1@NODE1.HP.COM
host/node1.hp.com@NODE1.HP.COM
host/node1@NODE1.HP.COM
kadmin/admin@NODE1.HP.COM
kadmin/changepw@NODE1.HP.COM
kadmin/history@NODE1.HP.COM
krbtgt/NODE1.HP.COM@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
2.7 Configuring HP TCP/IP Services for OpenVMS SSH with Kerberos

Using Kerberos with TCP/IP SSH for OpenVMS, you can authenticate your SSH connections between OpenVMS systems.

The minimum version of TCP/IP Services for OpenVMS necessary for Kerberized SSH is Version 5.6.

To "Kerberize" your SSH connections, perform the following steps.

1. Install and configure TCP/IP for OpenVMS Services Version 5.6 or higher.
2. Install and configure Kerberos for OpenVMS.
   
   If you have already installed OpenVMS Version 7.3-2 or higher, Kerberos is part of the OpenVMS installation procedure. If you have an earlier version of OpenVMS installed, you can download the Kerberos for OpenVMS PCSI kit from the Kerberos web site at http://h71000.www7.hp.com/openvms/products/kerberos/

3. Shut down Kerberos, if it is running, by entering the following command:

   $ @SYS$STARTUP:KRB$SHUTDOWN

4. Configure TCP/IP Services for OpenVMS by entering the following command:

   $ @SYS$STARTUP:TCPIP$CONFIG

5. Select #2, Client components, from the TCP/IP Configuration Menu:
HP TCP/IP Services for OpenVMS Configuration Menu

Configuration options:

1 - Core environment
2 - Client components
3 - Server components
4 - Optional components
5 - Shutdown HP TCP/IP Services for OpenVMS
6 - Startup HP TCP/IP Services for OpenVMS
7 - Run tests
A  - Configure options 1 - 4
[E] - Exit configuration procedure

Enter configuration option: 2

6. Ensure that the SSH Client and Server services are enabled. Select #7, SSH Client, from the TCP/IP Configuration Menu:

HP TCP/IP Services for OpenVMS Client Components Configuration Menu

Configuration options:

1 - DHCP Client Disabled Stopped
2 - FTP Client Enabled Started
3 - NFS Client Disabled Stopped
4 - REXEC and RSH Enabled Started
5 - RLOGIN Enabled Started
6 - SMTP Disabled Stopped
7 - SSH Client Disabled Stopped
8 - TELNET Enabled Started
9 - TELNETSYM Disabled Stopped

A  - Configure options 1 - 9
[E] - Exit menu

Enter configuration option: 7

7. Select #2, Enable service on this node, from the TCP/IP Configuration Menu. Type YES when it asks if you want to configure the SSH SERVER. If SSH is already enabled, skip to step 9.

SSH CLIENT configuration options:

1 - Enable service on all nodes
2 - Enable service on this node
3 - Stop service on this node

[E] - Exit SSH_CLIENT configuration

Enter configuration option: 2

The SSH SERVER is enabled.

* Do you want to configure SSH SERVER [NO]: YES

8. Select #2, Enable Service on this node, from the TCP/IP Configuration Menu. Press return to select the default or type YES to create a new default server host key.
SSH configuration options:

1 - Enable service on all nodes
2 - Enable service on this node
3 - Stop service on this node

[E] - Exit SSH configuration

Enter configuration option: 2
* Create a new default server host key? [YES]: YES
Creating private key file: TCPIP$SSHDEVICE:[TCPIP$SSH.SSH2]HOSTKEY
Creating public key file: TCPIP$SSHDEVICE:[TCPIP$SSH.SSH2]HOSTKEY.PUB

9. Select Exit twice to exit from each submenu of the TCP/IP Configuration Menu.

10. If the system asks if you want to start SSH now, answer NO.

The following services are enabled but not started:

SSH, SSH_CLIENT

* Start these services now? [N] NO

You may start services individually with:

@SYS$STARTUP:TCPIP$<service>_STARTUP.COM

11. If SSH is not already running, manually start the SSH client and server by entering the following commands:

$ @SYS$STARTUP:TCPIP$SSH_STARTUP.COM
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SSHD2.EXE installed
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SFTP-SERVER2.EXE installed
%TCPIP-I-INFO, logical names created
%TCPIP-S-STARTDONE, TCPIP$SSH startup completed

$ @SYS$STARTUP:TCPIP$ssh_client_STARTUP.COM
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SCP2.EXE installed
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SFTP2.EXE installed
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SSH-ADD2.EXE installed
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SSH-AGENT2.EXE installed
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SSH-KEYGEN2.EXE installed
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SSH-SIGNER2.EXE installed
%TCPIP-I-INFO, image SYSSYSTEM:TCPIP$SSH_SSH2.EXE installed
%TCPIP-I-INFO, logical names created
%TCPIP-S-STARTDONE, TCPIP$SSH_CLIENT startup completed

12. Start Kerberos by entering the following command:

$ @SYS$STARTUP:KRB$STARTUP

13. Verify that the SSH service is enabled by entering the following command:

$ TPCIP SHOW SERV

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
<th>Proto</th>
<th>Process</th>
<th>Address</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td>21</td>
<td>TCP</td>
<td>TCPIP$FTP</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
<tr>
<td>REXEC</td>
<td>512</td>
<td>TCP</td>
<td>TCPIP$REXEC</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
14. Modify the following SSH configuration files to enable the Kerberos authentication methods:

SYS$SYSDEVICE:[000000.TCPIP$SSH.SSH2]
  SSH2_CONFIG.           (SSH client)
  SSHD2_CONFIG.           (SSH server)

In each file, under the 'Authentication' section, you must add the Kerberos authentication methods you would like to use. Following is an example that uses all three methods, plus the regular methods. Make sure you indent and space as the example in the file shows:

```
AllowedAuthentications      gssapi-with-mic, kerberos-2@ssh.com,
                            kerberos-tgt-2@ssh.com, publickey,
                            password, hostbased
```

You should only have one AllowedAuthentications line uncommented. If there are others that are uncommented, comment them out with a # sign as shown below:

```
#  AllowedAuthentications       publickey, keyboard-interactive, password
```

15. Add the following lines to SYS$MANAGER:SYSTARTUP_VMS.COM to install the 32-bit Kerberos images at boot time. They are needed for the Kerberos-based functionality with SSH:

```
$ INSTALL CREATE SYS$SHARE:KRB$RTL32.EXE/OPEN/HEADER_RESIDENT/SHARED
$ INSTALL CREATE SYS$SHARE:GSS$RTL32.EXE/OPEN/HEADER_RESIDENT/SHARE
```

16. If you are using TCP/IP Version 5.6 and Kerberos Version 2.1 and want to use the gssapi-with-mic authentication method with SSH, you must define the following system logical:

```
$ DEFINE/SYSTEM TCPIP$SSH_KRBRTL_HACK 1
```

17. Set up the Kerberos symbols, if you have not already done so. Add the following command to the SYS$MANAGER:SYLOGIN.COM file.

```
$ @SYS$MANAGER:KRB$SYMBOLS
```

The following steps should be performed by each user who will use Kerberized SSH.

A. Log into the OpenVMS system.

```
Welcome to OpenVMS (TM) Alpha Operating System, Version 8.3
```

Username: user1
Password:

B. Perform a kinit with the principal name that matches the OpenVMS username. To do so, enter one of the following commands at the DCL prompt each time you start a Kerberized application, such as TCP/IP Services for OpenVMS SSH. You are then prompted for the password associated with the principal. (The -f is required for the kerberos-tgt-2 authentication method.)

```
$ kinit -f "USER1"
password for user1@NODE1.HP.COM

$ kinit "USER1"
password for user1@NODE1.HP.COM
```

C. Enter the SSH command specifying the Kerberos authentication method to use and the hostname as follows:
$ ssh -o "AllowedAuthentications gssapi-with-mic" node1
Authentication successful.
Welcome to OpenVMS (TM) Operating System, Version 8.3

$ ssh -o "AllowedAuthentications kerberos-2@ssh.com" node1
Authentication successful.
Welcome to OpenVMS (TM) Operating System, Version 8.3

$ ssh -o "AllowedAuthentications kerberos-tgt-2@ssh.com" node1
Authentication successful.
Welcome to OpenVMS (TM) Operating System, Version 8.3

D. See the HP TCP/IP Services for OpenVMS Guide to SSH for more information about configuring SSH and troubleshooting.

---

2.8 Configuring HP TCP/IP Services for OpenVMS Telnet with Kerberos

Using Kerberos with TCP/IP KTELNET for OpenVMS, you can authenticate your Telnet connections between OpenVMS systems.

The minimum version of TCP/IP Services for OpenVMS necessary for Kerberized Telnet is Version 5.3. If you are using a version of TCP/IP Services for OpenVMS prior to Version 5.5, you must download the Kerberized Telnet client (TCPIP$TELNET.EXE) and server (TCPIP$TELNET_SERVER.EXE) kits from http://h71000.www7.hp.com/openvms/products/kerberos/

**NOTE**
If you download the Telnet client and server, you must copy TCPIP$TELNET.EXE and TCPIP$TELNET_SERVER.EXE to SYS$COMMON:[SYSEXE].

You do not need to run these files directly. They are executed when you first run Telnet after following the instructions below.

**To "Kerberize" your Telnet connections, perform the following steps.**

1. Install and configure TCP/IP for OpenVMS Services Version 5.3 or higher.
2. Install and configure Kerberos for OpenVMS. If you have already installed OpenVMS Version 7.3-2 or higher, Kerberos is part of the OpenVMS installation procedure. If you have an earlier version of OpenVMS installed, you can download the Kerberos for OpenVMS PCSI kit from the Kerberos web site at http://h71000.www7.hp.com/openvms/products/kerberos/
3. Shut down Kerberos, if it is running, by entering the following command:
   
   $ SYS$STARTUP:KRBS$SHUTDOWN
4. Configure TCP/IP Services for OpenVMS by entering the following command:
   
   $ @SYS$STARTUP:TCPIP$CONFIG
5. Select #2, Client components, from the TCP/IP Configuration Menu:

HP TCP/IP Services for OpenVMS Configuration Menu

Configuration options:

1 - Core environment
2 - Client components
3 - Server components
4 - Optional components
5 - Shutdown HP TCP/IP Services for OpenVMS
6 - Startup HP TCP/IP Services for OpenVMS
7 - Run tests

A - Configure options 1 - 4
[E] - Exit configuration procedure

Enter configuration option: 2

6. Ensure that the Telnet service is stopped. If Telnet is already stopped, skip to step 8. If Telnet is not currently stopped, select #8, Telnet, from the TCP/IP Configuration Menu:

HP TCP/IP Services for OpenVMS Client Components Configuration Menu

Configuration options:

1 - DHCP Client    Disabled Stopped
2 - FTP Client     Enabled Started
3 - NFS Client     Disabled Stopped
4 - REXEC and RSH Enabled Started
5 - RLOGIN        Enabled Started
6 - SMTP          Disabled Stopped
7 - SSH Client    Enabled Started
8 - TELNET        Enabled Started
9 - TELNETSYM     Disabled Stopped

A - Configure options 1 - 9
[E] - Exit menu

Enter configuration option: 8

NOTE You must stop the Telnet service before you can begin to configure Kerberized Telnet. Stopping the Telnet service disconnects current Telnet sessions.

7. Select #3, Stop service on this node, from the TCP/IP Configuration Menu:

TELNET configuration options:

1 - Enable service on all nodes
2 - Enable service on this node
3 - Stop service on this node

[E] - Exit TELNET configuration

Enter configuration option: 3
8. Select [E], Exit menu, from the TCP/IP Configuration Menu:

Configuration options:

1 - DHCP Client      Disabled Stopped
2 - FTP Client       Enabled Started
3 - NFS Client       Disabled Stopped
4 - REXEC and RSH   Enabled Started
5 - RLOGIN           Enabled Started
6 - SMTP             Disabled Stopped
7 - SSH Client       Enabled Started
8 - TELNET           Enabled Stopped
9 - TELNETSYM        Disabled Stopped

A - Configure options 1 - 9
[E] - Exit menu

Enter configuration option: E

9. Select #4, Optional components, from the TCP/IP Configuration Menu:

HP TCP/IP Services for OpenVMS Configuration Menu

Configuration options:

1 - Core environment
2 - Client components
3 - Server components
4 - Optional components
5 - Shutdown HP TCP/IP Services for OpenVMS
6 - Startup HP TCP/IP Services for OpenVMS
7 - Run tests

A - Configure options 1 - 4
[E] - Exit configuration procedure

Enter configuration option: 4

10. Select #4, Configure Kerberos Applications, from the TCP/IP Configuration Menu:

HP TCP/IP Services for OpenVMS Optional Components Configuration Menu

Configuration options:

1 - Configure PWIP Driver (for DECnet-Plus and PATHWORKS)
2 - Configure SRI QIO Interface (INET Driver)
3 - Set up Anonymous FTP Account and Directories
4 - Configure Kerberos Applications
5 - Configure failSAFE IP

A - Configure options 1 - 5
[E] - Exit menu

Enter configuration option: 4

11. Select #1, Add Kerberos for TELNET server, from the TCP/IP Configuration Menu:

Kerberos Applications Configuration Menu

TELNET Kerberos is not defined in the TCPIP$SERVICE database.
Configuration options:

1 - Add Kerberos for TELNET server
2 - Remove Kerberos for TELNET server
[E] - Exit menu

Enter configuration option: 1

12. Select Exit three times to exit from the submenus of the TCP/IP Configuration Menu.

13. If the system asks if you want to start Telnet now, answer NO.

The following services are enabled but not started:

TELNET

Start these services now? [N] NO

You may start services individually with:

@SYS$STARTUP:TCPIP$<service>_STARTUP.COM

14. Manually start Telnet by entering the following command:

$ @SYS$STARTUP:TCPIP$TELNET_STARTUP.COM

%TCPIP-I-INFO, image SYS$SYSTEM:TCPIP$TELNET_SERVER.EXE installed
%TCPIP-I-INFO, image SYS$SYSTEM:TCPIP$TELNET.EXE installed
%TCPIP-I-INFO, logical names created
%TCPIP-I-INFO, telnet service enabled
%TCPIP-I-INFO, telnet (kerberos) service enabled
%TCPIP-S-STARTDONE, TCPIP$TELNET startup completed

15. Start Kerberos by entering the following command:

$ @SYS$STARTUP:KRB$STARTUP

16. Verify that the Kerberos Telnet (KTELNET) service is enabled by entering the following command. (If KTELNET is disabled, you can enable it using the $ TCPIP ENABLE SERVICE KTELNET command.)

$ TCPIP SHOW SERV

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
<th>Proto</th>
<th>Process</th>
<th>Address</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td>21</td>
<td>TCP</td>
<td>TCPIP$FTP</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
<tr>
<td>KTELNET</td>
<td>2323</td>
<td>TCP</td>
<td>TCPIP$TELNET</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
<tr>
<td>REXEC</td>
<td>512</td>
<td>TCP</td>
<td>TCPIP$REXEC</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
<tr>
<td>RLOGIN</td>
<td>513</td>
<td>TCP</td>
<td>not defined</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
<tr>
<td>RSH</td>
<td>514</td>
<td>TCP</td>
<td>TCPIP$RSH</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
<tr>
<td>SSH</td>
<td>22</td>
<td>TCP</td>
<td>TCPIP$SSH</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
<tr>
<td>TELNET</td>
<td>23</td>
<td>TCP</td>
<td>not defined</td>
<td>0.0.0.0</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

17. Set up the Kerberos symbols, if you have not already done so. Add the following command to the SYS$MANAGER:SYLOGIN.COM file.

$ @SYS$MANAGER:KRB$SYMBOLS

The following steps should be performed by each user who will user Kerberized Telnet.

A. Log into the OpenVMS system.
Welcome to OpenVMS (TM) Alpha Operating System, Version 8.3

Username: user1
Password:

B. Perform a kinit with the principal name that matches the OpenVMS username. To do so, enter the following command at the DCL prompt each time you start a Kerberized application, such as TCP/IP Services for OpenVMS Telnet. You are then prompted for the password associated with the principal. (The -f denotes forwardable credentials.)

```
$kinit -f "USER1"
password for user1@node1.hp.com
```

C. Enter the TELNET/AUTH command specifying Kerberos port 2323 to start the TELNET session, as follows:

```
$kinit -f "USER1"
$ TELNET/AUTH NODE1 2323
TELNET-I-TRYING, Trying ... 1.2.3.4
%TELNET-I-SESSION, Session 01, host node1, port 2323
-TELNET-I-ESCAPE, Escape character is ^
[ Kerberos V5 accepts you as 'user1.NODE1.HP.COM'
]
```

D. Optionally, enter the TELNET/AUTH/FORW command specifying Kerberos port 2323 to forward credentials. (Note: Forwarding credentials to non-OpenVMS servers works properly, but there is currently a problem in forwarding credentials to OpenVMS servers. This will be corrected in a future TCP/IP Services for OpenVMS ECO kit.)

```
$ TELNET/AUTH/FORW NODE1 2323
TELNET-I-TRYING, Trying ... 1.2.3.4
%TELNET-I-SESSION, Session 01, host node1, port 2323
-TELNET-I-ESCAPE, Escape character is ^
[Kerberos V5 accepts you as 'user1@NODE1.HP.COM'
][ Kerberos V5 refuses authentication ]
```

E. If you are using Kerberized Telnet to a non-OpenVMS system, the default port of 23 should be specified. Port 2323 is only used when contacting a Kerberized Telnet server on an OpenVMS system. This is because Telnet on OpenVMS currently uses different servers for regular and Kerberized Telnet.

2.9 Configuring and Starting the Kerberos ACME Agent

HP OpenVMS Version 8.3 includes pre-production images for the Kerberos ACME agent. The Kerberos ACME agent is an addition to the existing Kerberos authentication provided by the Kerberos utilities. The Kerberos ACME provides functionality similar to the pam_krb5 utility on UNIX systems using Kerberos.

**NOTE** The images described in this section are "pre-production" images and are not qualified for production use. After additional rigorous production quality testing and qualification is completed, a maintenance update (ECO) will be made available to allow for production use deployments.

To use Kerberos with previous versions of OpenVMS, you needed to log in twice: once to log in to OpenVMS itself, and once to obtain Kerberos credentials. These steps worked with separate principal, or user, names, and with separate passwords.
With the Kerberos ACME agent, you can obtain your Kerberos credentials as part of the OpenVMS login process. The user authentication is processed against the Kerberos KDC database instead of against the OpenVMS User Authorization File (UAF).

After you install and configure Kerberos Version 3.0, perform the following steps to configure and start the Kerberos ACME agent.

1. **Install ACME Login** from a privileged account. OpenVMS Version 8.3 includes pre-production images for ACME Login. See the file SYS$HELP:ACME_DEV_README.TXT for information about installation and set up.

2. **Install the Kerberos persona extension** by entering the following commands:

   ```
   $ MCR SYSMAN
   SYSMAN> SYS_LOADABLE ADD/LOG KERBEROS KRB$ACME_KRB_PERSONA_EXT
   %SYSMAN-I-IMGADDED, added image KRB$ACME_KRB_PERSONA_EXT for product KERBEROS
   $ @SYS$UPDATE:VMS$SYSTEM_IMAGES.COM
   ```

3. **Reboot the system.** This is required one time only, after you have installed the Kerberos persona extension.

4. To start the Kerberos ACME agent automatically, edit the file SYS$MANAGER:ACME$START.COM to uncomment the following line:

   ```
   $! @SYS$STARTUP:KRB$STARTUP_KERBEROS_ACME
   ```

5. **Edit the file SYSTARTUP_VMS.COM** to include the following command after all dependent software is started:

   ```
   $ SET SERVER ACME/RESTART
   ```

6. **Create an OpenVMS account** with the EXTAUTH flag set.

7. **Create a Kerberos principal name** that exactly matches (including case) the OpenVMS account name created in step 6. Passwords do not need to match. For the Kerberos configuration, you can use either DCL or UNIX-style commands to create the principal.

   The first example below shows the DCL commands. The second example shows the UNIX-style commands. Both styles of commands are entered on an OpenVMS system.

   **DCL:**

   ```
   $ KERBEROS/ADMIN
   KerberosAdmin> login "SYSTEM/admin"
   Enter password:
   Authenticating as principal SYSTEM/admin with password.
   KerberosAdmin> list principal
   K/M@NODE1.HP.COM
   SYSTEM/admin@NODE1.HP.COM
   kadmin/admin@NODE1.HP.COM
   kadmin/changepw@NODE1.HP.COM
   kadmin/nodel@NODE1.HP.COM
   kadmin/history@NODE1.HP.COM
   krbtgt/NODE1.HP.COM@NODE1.HP.COM
   KerberosAdmin> create principal "ACMEUSER"
   Authenticating as principal SYSTEM/admin with password.
   WARNING: no policy specified for ACMEUSER@NODE1.HP.COM; defaulting to
   no policy
   Enter password for principal "ACMEUSER@NODE1.HP.COM":
   Re-enter password for principal "ACMEUSER@NODE1.HP.COM":
   ```
Principal “ACMEUSER@NODE1.HP.COM” created.
KerberosAdmin> list principal
Authenticating as principal SYSTEM/admin with password.
K/M@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
ACMEUSER@NODE1.HP.COM
kadmin/admin@NODE1.HP.COM
kadmin/changepw@NODE1.HP.COM
kadmin/node1@NODE1.HP.COM
kadmin/history@NODE1.HP.COM
krbtgt/NODE1.HP.COM@NODE1.HP.COM

UNIX:

$ kinit “SYSTEM/admin”
Password for SYSTEM/admin@NODE1.HP.COM:

$ kadmin
Authenticating as principal SYSTEM/admin@NODE1.HP.COM with password.
Enter password:
KADMIN: listprincs
K/M@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
kadmin/admin@NODE1.HP.COM
kadmin/changepw@NODE1.HP.COM
kadmin/node1@NODE1.HP.COM
kadmin/history@NODE1.HP.COM
krbtgt/NODE1.HP.COM@NODE1.HP.COM
KADMIN: addprinc “ACMEUSER”
WARNING: no policy specified for ACMEUSER@NODE1.HP.COM; defaulting to no policy
Enter password for principal “ACMEUSER@NODE1.HP.COM”:
Re-enter password for principal “ACMEUSER@NODE1.HP.COM”:
Principal “ACMEUSER@NODE1.HP.COM” created.
KADMIN: listprincs
K/M@NODE1.HP.COM
SYSTEM/admin@NODE1.HP.COM
USER1@NODE1.HP.COM
kadmin/admin@NODE1.HP.COM
kadmin/changepw@NODE1.HP.COM
kadmin/node1@NODE1.HP.COM
kadmin/history@NODE1.HP.COM
krbtgt/NODE1.HP.COM@NODE1.HP.COM

8. SET HOST or Telnet to the system on which you installed the ACME Agent and the Kerberos persona extension in steps 1 and 2. Enter one of the following commands:

$ TELNET NODE1

or

$ SET HOST NODE1

9. Enter the username and password. You must enclose the username in quotes so that the case of the username is preserved. For example:

Welcome to OpenVMS (TM) Alpha Operating System, Version 8.3

Username: “ACMEUSER”
Password:

**** Logon Message from ACME_KRB_DOI ACME Agent ***
The logon message indicates that you successfully obtained your Kerberos credentials as part of the OpenVMS login process.
3 Kerberos Client and Administrative Programs

In addition to the Kerberos database and Key Distribution Center, there are a number of user and administrative programs that allow interaction with Kerberos. This chapter will detail the use of those programs.

The Kerberos user client programs include the following:

- kinit – Obtains a Kerberos ticket-granting ticket
- klist – Lists cached Kerberos tickets
- kdestroy – Destroys Kerberos tickets
- kpasswd – Changes a user's Kerberos password

The Kerberos administrative client programs include the following:

- kadmin and kadmin_local – Administers the Kerberos database
- kdb5_util – Dumps and restores the Kerberos database
- ktutil – Reads, writes, or edits entries in a Kerberos V5 keytab or V4 srvtab file
- kprop – Propagates the master KDC database to slave KDCs

The symbols for these programs are defined by SYS$MANAGER:KRB$SYMBOLS.COM.

On OpenVMS, these programs are located in the system directory and are prefaced by KRB$; for example, SYS$SYSTEM:KRB$KINIT.EXE.

NOTE All options for the client programs are case sensitive. Uppercase options should be enclosed in double quotation marks. For example:

$ kinit "-R"

3.1 User Client Programs

This section describes the user client programs kinit, klist, kdestroy, and kpasswd.

3.1.1 kinit

The kinit program allows the user to obtain and cache a Kerberos ticket-granting ticket. A Kerberos principal name must have already been created for the user, or another pre-existing principal must be specified.
The `kinit` program optionally uses the logical name `KRB5CCNAME` to specify the location and name of the credentials (ticket) cache. The default location for the credentials cache is in the `[/KRB.<nodename>]/` subdirectory of the user’s login directory. The default name of the credentials cache is `KRB5CC_xxxxxx.;` where `xxxxxx` is a randomly generated numeric string.

**SYNOPSIS**

```
```

**OPTIONS**

- `-5`  
  Get Kerberos 5 tickets, overriding the default built-in behavior. This option may be used with `-4`.

- `-4`  
  Get Kerberos 4 tickets, overriding the default built-in behavior. This option may be used with `-5`.

- `-V`  
  Display verbose output.

- `-l lifetime`  
  Request a ticket whose lifetime is specified by `lifetime`. The value for `lifetime` must be followed immediately by one of the following delimiters:

  - `s` seconds
  - `m` minutes
  - `h` hours
  - `d` days

  For example:

  ```
kinit -l 90m
  ```

  You cannot mix units; a value of `30h30m` will result in an error.

  If the `-l` option is not specified, the default ticket lifetime (configured by each site) is used. Specifying a ticket lifetime longer than the maximum ticket lifetime (configured by each site) results in a ticket with the maximum lifetime.

- `-s start_time`  
  Request a postdated ticket, valid starting at `start_time`. Postdated tickets are issued with the invalid flag set, and need to be fed back to the KDC before use.

- `-r renewable_life`  
  Request renewable tickets, with a total lifetime of `renewable_life`. The duration is the same format as the `-l` option, with the same delimiters.

  (Not applicable to Kerberos 4.)

- `-f`  
  Request tickets that can be forwarded to another system. (Not applicable to Kerberos 4.)

- `-F`  
  Do not request forwardable tickets. (Not applicable to Kerberos 4.)

- `-p`  
  Request proxiable tickets. (Not applicable to Kerberos 4.)

- `-P`  
  Do not request proxiable tickets. (Not applicable to Kerberos 4.)

- `-A`  
  Request address-less tickets. (Not applicable to Kerberos 4.)
-v
Request that the ticket granting ticket in the cache (with the invalid option set) be passed to the KDC for validation. If the ticket is within its requested time range, the cache is replaced with the validated ticket. (Not applicable to Kerberos 4.)

-R
Request renewal of the ticket-granting ticket. Note that an expired ticket cannot be renewed, even if the ticket is still within its renewable life. When using this option with Kerberos 4, the KDC must support Kerberos 5 to Kerberos 4 ticket conversion.

-k [t keytab_file]
Request a host ticket, obtained from a key in the local host’s keytab file. The name and location of the keytab file may be specified with the -t keytab_file option; otherwise the default name and location will be used. When using this option with Kerberos 4, the KDC must support Kerberos 5 to Kerberos 4 ticket conversion.

-c cache_name
Use cache_name as the credentials (ticket) cache name and location; if this option is not used, the default cache name and location are used.

The default credentials cache may vary between systems. If the KRB5CCNAME logical name is set, its value is used to name the default ticket cache. Any existing contents of the cache are destroyed by kinit. (Not applicable to Kerberos 4.)

-S service_name
Specify an alternate service name to use when getting initial tickets.

### 3.1.2 klist

The klist program allows the user to display information about their cached Kerberos tickets. (Applicable to Kerberos 5, or to Kerberos 4 ticket conversion if you use both Kerberos 5 and Kerberos 4 with a KDC that supports Kerberos 5.)

**SYNOPSIS**

```plaintext
   [ cache_name | keytab_name ]
```

**OPTIONS**

-5
List Kerberos 5 credentials. This overrides whatever the default built-in behavior may be. This option may be used with -4.

-4
List Kerberos 4 credentials. This overrides whatever the default built-in behavior may be. This option may be used with -5.

-e
Display the encryption types of the session key and the ticket for each credential in the credential cache, or each key in the keytab file.

-c
List the tickets held in a credentials cache. This is the default if neither -c nor -k is specified.

-f
Show the options present in the credentials. Possible options are as follows:

- A Pre-authenticated
- a anonymous
- D postDateable
Kerberos Client and Administrative Programs

User Client Programs

- d postdated
- F Forwardable
- f forwarded
- H Hardware authenticated
- I Initial
- i invalid
- O OK as delegate
- P Proxiable
- p proxy
- R Renewable
- T Transit policy checked

-s Cause klist to run silently (produce no output) but to still set the exit status according to whether it finds the credential cache. The exit status is $SS$_NORMAL if klist finds a credentials cache.

-a Display list of addresses in credentials.

-n Show numeric addresses instead of reverse-resolving addresses.

-k List the keys held in a keytab file.

-t Display the time entry timestamps for each keytab entry in the keytab file.

-K Display the value of the encryption key in each keytab entry in the keytab file.

If cache_name or keytab_name is not specified, klist will display the credentials in the default credentials cache or keytab file as appropriate. If the KRB5CCNAME logical name is set, its value will be used to name the default ticket cache.

3.1.3 kdestroy

The kdestroy program destroys the user's active Kerberos authorization tickets by writing zeros to the specified credentials cache that contains them. If the credentials cache is not specified, the default credentials cache is destroyed. The default behavior is to destroy both Kerberos 5 and Kerberos 4 credentials.

SYNOPSIS

kdestroy [-5] [-4] [-q] [-c cache_name]

OPTIONS

-5 Destroy Kerberos 5 credentials. This option may be used with –4.

-4 Destroy Kerberos 4 credentials. This option may be used with –5.

-q Quiet mode. Normally, kdestroy beeps if it fails to destroy the user's tickets, in addition to issuing an error message. The –q option suppresses the beep, and only an error is issued.

-c cache_name Use cache_name as the credentials (ticket) cache name and location. If this option is not used, the default cache name and location are used.
If the **KRB5CCNAME** logical name is set, its value is used to name the default ticket cache.

HP recommends that you place the `kdestroy` command in a logout command file, so that your tickets are destroyed automatically when you log out.

### 3.1.4 kpasswd

The **kpasswd** program is used to change a Kerberos principal's password. The **kpasswd** program prompts for the current Kerberos password, which is used to obtain a `changepw` ticket from the KDC for the user's Kerberos realm. If **kpasswd** successfully obtains the `changepw` ticket, the user is prompted twice for the new password, and the password is changed.

If the principal is governed by a policy that specifies the length or number of character classes required in the new password, the new password must conform to the policy. (The five-character classes are: lowercase, uppercase, numbers, punctuation, and all other characters.)

**SYNOPSIS**

```plaintext
kpasswd
```

**OPTIONS**

- **principal**
  
  Change the password for the Kerberos principal specified by `principal`. Otherwise, the principal is derived from the identity of the user invoking the **kpasswd** command.

### 3.2 Administrative Client Programs

This section describes the administrative utilities **kadmin**, **kadmin_local**, **kdb5_util**, and **kprop**.

#### 3.2.1 kadmin and kadmin_local

The **kadmin** program allows the Kerberos administrator to make changes to the Kerberos database. The **kadmin** program provides for the maintenance of Kerberos principals, policies, and service key tables (keytabs). It exists as both a Kerberos client (**kadmin**), using Kerberos authentication and an RPC to operate securely from anywhere on the network, and as a local client (**kadmin_local**), intended to run directly on the KDC without Kerberos authentication.

**SYNOPSIS**

```plaintext
kadmin [-r realm] [-p principal] [-w password] [-q query]
[-s admin_server[:port]] [|-c credentials_cache] |
[-k keytab]]
```

```plaintext
kadmin_local [-d dbname] [-e “enc:salt ...”] [-m]
```

**Options**

- **-r realm**
  
  Use `realm` as the default Kerberos realm for the database.
**Kerberos Client and Administrative Programs**

**Administrative Client Programs**

-p principal

Use the Kerberos principal `principal` to authenticate to Kerberos. If this option is not given, `kadmin` will append `admin` to either the primary principal name or to the username of the current process.

-w password

Use `password` as the password instead of prompting for one. **Caution:** Placing the password for a Kerberos principal with administrative access into a command file can be dangerous if unauthorized users gain read access to the file.

-q query

Pass the string `query` directly to `kadmin`. This is useful for writing command procedures that pass specific queries to `kadmin`.

-s admin_server[:port]

Use `admin_server` as the KDC to contact. Optionally specify the TCP/IP port to use for communication.

-c credentials_cache

Use `credentials_cache` as the credentials cache. The credentials cache should contain a service ticket for the `kadmin/admin` service, which can be acquired with the `kinit` program. If this option is not specified, `kadmin` requests a new service ticket from the KDC and stores it in its own temporary cache.

-k keytab

Use the keytab `keytab` to decrypt the KDC response instead of prompting for a password on the terminal. In this case, the principal will be `host/hostname`.

-d dbname

This option is valid for `kadmin_local` only. Specify the filename of the KDC database.

-e "enc:salt..."

This option is valid for `kadmin_local` only. It sets the list of cryptosystem and salt types to be used for any new keys created. Available types include `des3-cbc-sha1:normal`, `des-cbc-crc:normal`, and `des-cbc-crc:v4`.

-m

This option is valid for `kadmin_local` only. Specify the KDC database master key.

### 3.2.2 kdb5_util

The `kdb5_util` program provides a way for the Kerberos administrator to create, delete, load, or dump a Kerberos database. It also includes a command to stash a copy of the master database key in a file on a KDC, so that the KDC can authenticate itself to the `kadmind` and `krb5kdc` daemons at boot time.

**SYNOPSIS**

```
```

**OPTIONS**

-r realm

Use `realm` as the default Kerberos realm for the database.

-d dbname

Specify the filename of the KDC database.

-k mkeytype

Specify the encryption type to use from the list of supported `mtypes` in `KDC.CONF`.

-M mkeyname

Specify the master key name.
-sf stashfilename

Specify the file that stores the master key. If you specify this file, you are not prompted for the master key.

-m

Specify the KDC database master key.

command

The kdb5_util command can be one of the following:

ark [-e etype_list] principal

Add a random key for a Kerberos 5 database entry principal. This assumes the max key version number. As a side effect, all old keys older than the maximum key version number are deleted.

- e etype_list

Specify the key salt to use for the random key.

create [-s]

Create a new Kerberos database. If you specify the -s option, kdb5_util stashes a copy of the master key in a stash file.

destroy [-f]

Destroy the existing Kerberos database. If you do not specify the -f option, you are prompted with "are you sure?" before the database is destroyed.


Dump a Kerberos database to a file.

-old

Cause the dump file to be Kerberos 5 Beta 5 and earlier dump format (kdb5_edit load_dump version 2.0).

-b6

Cause the dump file to be Kerberos 5 Beta 6 format ("kdb5_edit load_dump version 3.0").

-ov

Cause the dump to be in ovsec_adm_export format.

-verbose

Cause the name of each principal and policy to be printed as it is loaded.

-mkey_convert

Change master key as part of dump.

-new_mkey_file mkey file

Get master key from file mkey_file.

-rev

Dump in reverse order.
**Kerberos Client and Administrative Programs**

**Administrative Client Programs**

- **re-**
  Do recursive descent tree traversal of database instead of using previous/next pointers.

file

File name of the dump file to output.

[princs]

Principal name to be dumped.

dump_v4 file

Dump a Kerberos database to a file in Kerberos V4 format.

file

File name of the dump file to output.


Restore a Kerberos database dump from a file, specified by file.

-old

Requires the dump to be in the Kerberos 5 Beta 5 and earlier dump format (kdb5_edit load_dump v2.0).

-b6

Require the dump to be in the Kerberos 5 Beta 6 format (kdb5_edit load_dump v3.0).

-ov

Require the dump to be in ovsec_adm_export format

-verbose

Cause the name of each principal and policy to be printed as it is dumped.

-update

Cause records from the dump file to be updated in or added to the existing database.

file

File name of the dump file to load.


Restore a Kerberos database dump from a Kerberos V4 format dump file (specified by input).

-t

Allow modification of an existing database. If you do not specify -t, the load will abort if the database exists.

-n
Read the Kerberos V4 master key from the key file.

-\(v\)

Cause the name of each principal and policy to be printed as it is loaded.

-\(K\)

Prompt for the Kerberos V5 database master password.

-\(s\) \texttt{stashfile}

Specify the location of the Kerberos V4 master key file.

\texttt{inputfile}

Filename of the V4 dump file to load.

\texttt{stash [-f keyfile]}

Create a stash file, which allows a KDC to authenticate itself to the database programs \texttt{kadmin}, \texttt{kadmind}, \texttt{krb5kdc}, and \texttt{kdb5_util}. If the \(-f\) option is not specified, \texttt{kdb5_util} stashes the key in the file specified in the \texttt{KRB$ROOT:[KRB5KDC]KDC.CONF} file.

3.2.3 \texttt{ktutil}

The \texttt{ktutil} program invokes a menu from which an administrator can read, write, or edit entries in a Kerberos V5 \texttt{keytab} or V4 \texttt{srvtab} file.

**SYNOPSIS**

\texttt{ktutil}

\texttt{command}

The command on the \texttt{ktutil} menu can be one of the following:

\texttt{clear_list, clear}

Clear the current key list.

\texttt{read_kt, rkt}

Read a \texttt{krb5} keytab into the current keylist.

\texttt{read_st, rst}

Read a \texttt{krb4 srvtab} into the current keylist.

\texttt{write_kt, wkt}

Write the current keylist to a \texttt{krb5} keytab.

\texttt{write_st, wst}

Write the current keylist to a \texttt{krb4 srvtab}.  


Kerberos Client and Administrative Programs

Administrative Client Programs

`add_entry`, `addent`
Add an entry to the current keylist.

`delete_entry`, `delent`
Delete an entry from the current keylist.

`list`, `l`
List the current keylist.

`list_requests`, `lr`, `?`
List available requests.

`quit`, `exit`, `q`
Exit program.

### 3.2.4 kprop

The `kprop` program propagates the master KDC database to slave KDCs.

The following sections describe the procedure you should use to propagate your master KDC database. This procedure involves performing steps first on the master system, then the slave system, and back and forth again until finishing with the master system.

In the following procedure, the steps are numbered M1, M2, and so on for the master KDC server, and S1, S2 and so on for the slave KDC server.

Kerberos must be installed on both the master and slave systems.

**PROCEDURE**

#### 3.2.4.1 Step 1: Configure the Master KDC Server for Propagation

**M1.** On the master KDC server, enter the following command:

```bash
$ @SYS$STARTUP:KRB$CONFIGURE
```

**M2.** Set up the client.

**M3.** Set up the server.

**M4.** Exit the `KRB$CONFIGURE.COM` file.

**M5.** If you added additional `USER/admin` principals during your configuration (other than your first admin principal), add them to `KRB$ROOT:[KRB5KDC]KADM5.ACL`.

**M6.** Add your anticipated slave hosts to `KRB$ROOT:[ETC]KRB5.CONF` under the `realms` tag using a `kdc` tag as follows:

```plaintext
USER/admin@REALM
kdc = nodename.domain:88
```

**M7.** To create `KRB$ROOT:[BIN]KRB$KPROP.DAT` from the template file `KRB$KPROP_DAT.TEMPLATE`, copy `KRB$KPROP_DAT.TEMPLATE` to `KRB$KPROP.DAT`, and edit `KRB$KPROP.DAT` as follows:

a. Comment out the example node name lines with a `#` sign.
b. Add all of your slave node names either as just the simple node name or as fully qualified node names that include their respective domain names. Be consistent in the naming method you choose. It is safest to use the node name form that is used to define your node names in your local TCP/IP host setting. If you use DNS to manage your local host lookups, you will need to use fully qualified node and domain name strings.

If you specify local host names, know the form of the node name you use, define all propagation node names that way in the local TCP/IP host database, and enter these propagation node names in the form that they are locally defined.

Try to define all propagation nodes in your local TCP/IP hosts database, or leave them all defined in DNS and not in your local database. If you see client not found errors during propagation, review your node name definitions and the form that you have in the local TCP/IP database.

c. The KRB$KPROP.DAT file is simply a data file that is read by the kprop command file to see where database propagation is performed. Make sure you do not include the local server node name in this data file. The propagation server does not need its own data propagated to itself.

d. You need only perform step M7 on those nodes that might act as the master KDC server at some future point, and need to have master database changes propagated to them.

M8. Create the KRB$ROOT:\[KRB5KDC\]KPROPD.ACL file as follows. There is no template for this file. This file defines the names of the hosts that will be involved in propagation and includes the master server entity.

a. Edit KRB$ROOT:\[KRB5KDC\]KPROPD.ACL to add each slave KDC host/name keytab entry that will be created in Step M11.

The form depends on how your node names are defined in TCP/IP. You can use either of the following forms. The @REALM portion is required.

  host/yournode@REALM
  host/yournode.yourdomain@REALM

b. If your local TCP/IP database defines the node names, the form of your node name in Step M8a must match that of your TCP/IP database

c. Be sure to include the host/entry for your master KDC.

M9. Start your master server and run KADMIN.

**NOTE**

In steps M10 and M11, it is critical that the node names are in the same form as your local TCP/IP node name. You can use either simple node names or fully qualified DNS node names, as long as you are consistent.

M10. Add the host/principals with the following commands:

  addprinc -randkey host/yourmasternode
  addprinc -randkey host/yourslavenode

M11. Add/export the host/keytabs with the following commands:

  ktadd host/yourmasternode@REALM
  ktadd host/yourslavenode@REALM
NOTE
The @REALM part of this file name is important and must match the REALM entered into KPROPD.ACL in step M8.

M12. Restart your master KDC server using the latest configuration.

3.2.4.2 Step 2: Configure the Slave KDC Servers for Propagation

After you configure the master server, perform the following steps to configure each slave KDC server.

S1. To configure your slave KDC client, enter the master KDC server name when asked where the master KDC server resides. Do not use your local node name.

S2. Set up your slave KDC server by entering the following command:

   $ @SYS$STARTUP:KRB$CONFIGURE

Note the following:

- Your KDC node name is your local node, not the master KDC node name.
- Specify SLAVE_KDC, if it is not the default.
- Add a local admin principal. (This will not be used.)
- Accept the defaults for the remaining questions.

S3. Exit the configuration file and perform step M7 from the previous section only if, in the future, you may use this slave KDC as a master KDC server. Otherwise, go to step S4.

S4. Perform Step 1, M8 on your slave KDC node. You can copy the file from the server or edit a new file using the same host/entry information. This step is required for propagation.

S5. Export the master server's host/keytabs to the local KDC slave server keytab file. Because the slave server is configured as a client in the master KDC, you can kinit as the master KDC server's admin and run kadmin to extract the server's keytabs as shown in Step 1, M11. This will create your local keytab file with the MASTER KDC server keytab information. Issue a listprincs command and then ktadd the host principals.

S6. Edit KRB$ROOT:[KRB5KDC]KRB$ROLE.DAT. Change the second data line from a zero to a one (0 to 1), and save the file. This tells KRB$CONFIGURE that the KRB$KPROPD.EXE daemon must be started when the slave server is started.

S7. Edit KRB$ROOT:[ETC]KRB5.CONF and add the slave and master KDC nodes under the realms tag, if they do not exist. Here, you can safely specify fully qualified node names with their domain names as follows:

   kdc = yourmasternode.yourdomain:88
   kdc = yourslavenode.yourdomain:88

Make sure the record format for KRB5.CONF and KPROPD.ACL is STREAM_LF.

CAUTION Do not start the slave server yet.

3.2.4.3 Step 3: Complete the Configuration of the Master KDC Server

Perform the following steps on the master server.
M13. Run `kadmin` and re-export only the master's `host/keytab` as in Step 1, M11. Because this keytab was exported on one or more slaves, the key version number is now greater than when this keytab was originally exported, and the slave KDCs will not be able to authenticate to the master KDC with a lower key version number.

M14. In `kadmin`, enter the following command:

```
ktadd host/yourmaster@REALM
```

**NOTE**
You may have to remove the host keytabs and principals and re-add them if the slave and master cannot agree on the key version numbers. This is an issue only with the master KDC keytab after keys are added to the slaves. This step does correct certain authentication problems.

M15. Restart the master server.

### 3.2.4.4  Step 4: Complete the Configuration of the Slave KDC Server

Perform the following steps on the slave server.

S8. Use `kinit` to get to your master server's `admin` account. This will refresh the master's host keytab on the local system and start the slave server in preparation for its first propagation from the master.

### 3.2.4.5  Step 5: Propagate the Master KDC Server to Each Configured Slave Server

Perform the following steps to complete the propagation procedure.

M16. Enter the following command:

```
@KRB$ROOT:[BIN]KRB$KPROP.COM
```

The `kprop` command procedure causes the following to occur:

a. The master server is stopped, the database dumped, the servers restarted, and a connection to each slave `kpropd` daemon is made in order to transfer the master database to the slave servers listed in `KRB$ROOT:[BIN]KRB$KPROP.DAT`.

b. The slave servers are stopped, the master KDC database is loaded, the slave servers are restarted, and a signal is sent to the master server that the propagation has successfully completed.

c. The master server produces a file called `SLAVE_DATATRANS_DAT_YOURSLAVENODE.LAST_PROP` that indicates that the propagation to the individual slave node has completed.

d. When propagation to each slave server completes, the `kpropd.exe` daemon exits. The next propagation can be done only after starting the `kpropd` daemon on each of the KDC slave servers. This is why `kpropd` should be a TCP/IP service. The TCP/IP system automatically starts the `kpropd` daemon for each slave server requested by the master server.
4 Kerberos Programming Concepts

This chapter provides an overview of programming with Kerberos on OpenVMS. Information in this chapter includes:

- An overview of building a Kerberos application on OpenVMS
- Descriptions of the Kerberos example programs

4.1 Overview of Building a Kerberos Application on OpenVMS

Kerberos programming on OpenVMS works much the same as on any other platform. The following sections indicate differences and important information.

4.1.1 Compiling a Kerberos Program on OpenVMS

When you compile your program, you will need to add the `/INCLUDE=KRB$ROOT:[INCLUDE]` qualifier to your compiler command line. For example:

```
$ cc/list/include=krb$root:[include]/prefix=all gss_client
```

4.1.2 Linking a Kerberos Program on OpenVMS

Kerberos on OpenVMS provides shareable libraries in both 64-bit and 32-bit formats. All Kerberos libraries can be found in `SYS$LIBRARY`.

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Bit Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS$RTL.EXE</td>
<td>64 bits</td>
</tr>
<tr>
<td>GSS$RTL32.EXE</td>
<td>32 bits</td>
</tr>
<tr>
<td>KRB$RTL.EXE</td>
<td>64 bits</td>
</tr>
<tr>
<td>KRB$RTL32.EXE</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

One of the `GSS$RTL*` libraries should be used when your program calls the GSS API. If the KRB5 API is called, then one of the `KRB$RTL*` libraries will need to be linked with the program.

Because Kerberos routines are located in shareable libraries, the use of a link options file is recommended. For details about using link options files, refer to the *HP OpenVMS Linker Utility Manual*. The Kerberos example programs described in this chapter provide examples of using link options files for Kerberos applications.
4.2 Kerberos Example Programs

This section describes the Kerberos example programs. Kerberos must be configured before any example program is run. For the configuration procedure, see Chapter 2.

The Kerberos example programs are found in SYS$COMMON:[SYSLIB.EXAMPLES.KERBEROS...].

The Kerberos example programs are divided between those examples that use DCL to build and those that use GMAKE to build.

4.2.1 DCL Example Programs

The SYS$COMMON:[SYSLIB.EXAMPLES.KERBEROS.DCL] directory in the Kerberos example directory tree contains the Version 1.0 example programs and build procedures. (No new examples were added to the DCL directory for Version 3.0.) These example programs are described in the following sections.

There are two DCL example programs, each of which has a client and server piece. Command procedures to build and help set up the example programs are provided, along with readme files specific to each example.

The examples should be built and run from a local build area or directory. The following table lists the DCL example programs and information about what aspect of Kerberos each program is attempting to convey.

<table>
<thead>
<tr>
<th>DCL Example Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_CLIENT and GSS_SERVER</td>
<td>GSSAPI example program</td>
</tr>
<tr>
<td>KRB_CLIENT and KRB_SERVER</td>
<td>KRB5 API example program</td>
</tr>
</tbody>
</table>

4.2.1.1 GSSAPI Example Program

The GSSAPI example program is a simple client/server program that authenticates using the GSSAPI.

To run the GSSAPI example client program, perform the following steps:

1. Create a Kerberos principal name of gss_sample/<node name>@<realm name> before this program is run.
2. Copy the GSS_*.* example files and the BUILD.COM and SETUP.COM files into a local build area, and then execute the BUILD command file as follows:

   $ COPY SYS$COMMON:[SYSLIB.EXAMPLES.KERBEROS.DCL]GSS*.* local_build_area
   $ COPY SYS$COMMON:[SYSLIB.EXAMPLES.KERBEROS.DCL]*.COM local_build_area
   $ SET DEF local_build_area
   $ @BUILD GSS

3. Execute the SETUP command file to define the necessary symbols to run the example.
4. Ensure that Kerberos has been initialized and started, and that the necessary Kerberos principal name has been created in the Kerberos database. The SETUP command file has additional information about creating the principal name.
5. Copy either GSS_CLIENT.EXE or GSS_SERVER.EXE to another node in the same Kerberos realm, along with the SETUP command file.
6. Start the client program and server programs using the symbols defined in SETUP.COM.
4.2.1.1 GSS_CLIENT

SYNOPSIS

gss_client [-p port] [message] [host] [service]

OPTIONS

-p port
Specifies the TCP/IP port to use for communications. If this argument is absent, port number 4444 is used.

message
Specifies the text message to pass between client and server.

host
Specifies the host system where the GSS_SERVER is located.

service

4.2.1.2 GSS_SERVER

SYNOPSIS

gss_server [-p port] [-l logfile] [service]

OPTIONS

-p port
Specifies the TCP/IP port to use for communications. If this argument is absent, port number 4444 is used.

-l logfile
Indicates that a logging file with the file name specified by logfile should be opened when the GSS_SERVER program is started.

service
Specifies the service name. If this argument is absent, gss_sample is used as the service name.

4.2.1.2 KRB5 API Example Program

The KRB5 example program is a simple client/server program that authenticates using the Kerberos API.

To run the KRB5 API example program, perform the following steps:

1. Create a Kerberos principal name of krb_sample/node name@realm name before this program is run.

2. Copy the KRB_*.* example files and the BUILD.COM and SETUP.COM files into a local build area, and then execute the BUILD command file as follows:

   $ COPY SYS$COMMON:[SYSHLP.EXAMPLES.KERBEROS.DCL]KRB*.* local_build_area
   $ COPY SYS$COMMON:[SYSHLP.EXAMPLES.KERBEROS.DCL]*.COM local_build_area
   $ SET DEF local_build_area
   $ @BUILD KRB

3. Execute the SETUP command file to define the necessary symbols to run the example.

4. Ensure that Kerberos has been initialized and started and that the necessary Kerberos principal name has been created in the Kerberos database. The SETUP command file has additional information about creating the principal name.
5. Copy either the KRB_CLIENT.EXE or KRB_SERVER.EXE to another node in the same Kerberos realm, along with the SETUP command file.

6. Start the client and server programs using the symbols defined in SETUP.COM.

### 4.2.1.2.1 KRB5_CLIENT

#### SYNOPSIS

```
krb5_client [-p port] [message] [host] [service]
```

#### OPTIONS

- `-p port`
  Specifies the TCP/IP port to use for communications. If this argument is absent, port number 4444 is used.

- `message`
  Specifies the text message to pass between client and server.

- `host`
  Specifies the host system where the KRB_SERVER is located.

- `service`
  Specifies the service name. If this argument is absent, krb_sample is used as the service name.

### 4.2.1.2.2 KRB5_SERVER

#### SYNOPSIS

```
krb_server [-p port] [-l logfile] [service]
```

#### OPTIONS

- `-p port`
  Specifies the TCP/IP port to use for communications. If this argument is absent, port number 4444 is used.

- `-l logfile`
  Indicates that a logging file with the file name specified by logfile should be opened when the KRB_SERVER program is started.

- `service`
  Specifies the service name. If this argument is absent, krb_sample is used as the service name.

### 4.2.2 GMAKE Example Programs

The SYS$COMMON:[SYSHLP.EXAMPLES.KERBEROS.GMAKE...] directory in the Kerberos example directory tree contains the examples that build with GMAKE.

#### 4.2.2.1 GMAKE.VMS Directory

The example programs in the SYS$COMMON:[SYSHLP.EXAMPLES.KERBEROS.GMAKE.VMS] subdirectory contain the original OpenVMS Kerberos Version 1.0 example programs (GSSAPI and krb5). These examples are built with GMAKE instead of DCL. These programs show you how the two GMAKE and DCL build processes compare using the same code base.
This build can produce the GSS and KRB example programs built against the 64-bit and 32-bit Kerberos and GSS libraries respectively. Both types of builds can be produced without directory conflict, and they can be run out of their respective build directories.

The server awaits a connection on a socket, receives a message from the client that it prints out, and then echoes back to the client. Run each program with "-?" to see the runtime options for the client and server.

### 4.2.2.2 GMAKE.MIT Directory

Four example programs are included in the `SYS$COMMON:[SYSLP.EXAMPLES.KERBEROS.GMAKE.MIT]` subdirectory.

Each of these examples builds against the 32-bit KRB and GSS runtime libraries. Because of the form of UNIX I/O functions that they use, the 64-bit Kerberos libraries cannot be used.

The following table lists the new GMAKE example programs found in `SYS$COMMON:[SYSLP.EXAMPLES.KERBEROS.GMAKE.MIT]` and information about what aspect of Kerberos each program is attempting to convey.

<table>
<thead>
<tr>
<th>GMAKE Example Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS-SAMPLE</td>
<td>32-bit based application that uses the 32-bit GSS$RTL32 library on Alpha, and the 32-bit implementations of the UNIX I/O library function calls</td>
</tr>
<tr>
<td>SAMPLE</td>
<td>Demonstration client/server application</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>UDP-based client and server application</td>
</tr>
<tr>
<td>USER_USER</td>
<td>Demonstrates a TCP/IP service name used to securely communicate between two network applications</td>
</tr>
</tbody>
</table>

### 4.2.2.3 GSS-SAMPLE Example Program

The `SYS$COMMON:[SYSLP.EXAMPLES.KERBEROS.GMAKE.MIT.GSS-SAMPLE]` subdirectory contains a `GSS-SAMPLE.README` file that describes in detail the function and operation of the `GSS-SAMPLE` program. It is a 32-bit based application that uses the 32-bit GSS$RTL32 library on Alpha. It also uses the 32-bit implementations of the UNIX I/O library function calls.

This directory also contains a sample GSSAPI client and server application. In addition to serving as an example of GSSAPI programming, this application is also intended to be a tool for testing the performance of GSSAPI implementations. Each time the client is invoked, it performs one or more exchanges with the server.

The client application can be used to simulate a variety of workloads on the server. It can serve as an example of how to create a performance application to test a new Kerberos GSSAPI based application of your own.

Several command line options can be used to define how the client will interact with the server. The `GSS-SAMPLE.README` file lists these options in detail. The following is a summary of `GSS-SAMPLE` options:

**SYNOPSIS**

```
```
OPTIONS

-d
Tells the client to delegate credentials to the server. For the Kerberos GSSAPI mechanism, this means that a forwardable TGT will be sent to the server, which will put it in its credential cache. You must have acquired your tickets with \texttt{kinit -f} for this to work.

-f
Tells the client that the \texttt{msg} argument is actually the name of a file whose contents should be used as the message.

-ccount
Specifies how many sessions the client should initiate with the server (the \textit{connection count}).

-mcount
Specifies how many times the message should be sent to the server in each session (the \textit{message count}).

-na
Tells the client not to do any authentication with the server. Implies \texttt{-nw, -nx and -nm}.

-nx
Tells the client not to encrypt messages.

-nw
Tells the client not to wrap messages. Implies \texttt{-nx}.

-nm
Tells the client not to ask the server to send back a cryptographic checksum (MIC).

4.2.2.4 SAMPLE Example Program
The \texttt{SYS$COMMON:[SYSHLP.EXAMPLES.KERBEROS.GMAKE.MIT.SAMPLE]} subdirectory contains the build for a server and a client called \texttt{sserver} and \texttt{sclient}, respectively, that are a simple demonstration client/server application. When \texttt{sclient} connects to \texttt{sserver}, it performs a Kerberos authentication, then \texttt{sserver} returns to \texttt{sclient} the Kerberos principal that was used for the Kerberos authentication. This example provides a good test that Kerberos has been successfully installed and configured on a machine.

The \texttt{sclient} and \texttt{sserver} images are built in separate directories, but the client and server are run from the top-level directory. There is a complete README file in the \texttt{sserver} directory that describes the detailed information for configuring and running these examples. You can get a fast start by simply running \texttt{SAMPLE\_SETUP.COM} in this directory for both the client and the server windows.

4.2.2.5 SIMPLE Example Program
The \texttt{SYS$COMMON:[SYSHLP.EXAMPLES.KERBEROS.GMAKE.MIT.SIMPLE]} subdirectory contains a UDP-based client and server example. It is similar to the original Version 1.0 \texttt{KRB\_CLIENT} and \texttt{KRB\_SERVER} examples, except that it uses UDP socket-based I/O. The server receives a message from the client and simply reports what it has received. The client reports that it successfully sent the data.
4.2.2.6 USER_USER Example Program

The SYS$COMMON:[SYSHLP.EXAMPLES.KERBEROS.GMAKE.MIT.USER_USER] subdirectory holds a client and a server example that can be used to see how a TCP/IP service name can be used to securely communicate between two network applications. It is similar to the original Version 1.0 KRB_CLIENT and KRB_SERVER examples, except that a TCP/IP service name is defined and used to tell the client the port number on which the server is listening. The client sends its data to the server and the server responds to the client with the message the client sent.
5 GSSAPI (Generic Security Services Application Programming Interface)

This chapter describes the C language bindings for the routines that make up the Generic Security Services Application Programming Interface (GSSAPI).

The GSSAPI provides security services to its callers, and is intended for implementation atop alternative underlying cryptographic mechanisms. In this manual, the underlying cryptographic mechanism is assumed to be Kerberos.

The GSSAPI allows a communicating application to authenticate the user associated with another application, to delegate rights to another application, and to apply security services such as confidentiality and integrity on a per-message basis.

There are four stages to using the GSSAPI:

- The application acquires a set of credentials with which it can prove its identity to other processes.
- A pair of communicating applications establish a joint security context using their credentials. The security context is a pair of GSSAPI data structures that contain shared state information.
- Per-message services are invoked to apply either integrity and data origin authentication, or confidentiality, integrity, and data authentication to application data.
- At the completion of a communications session, the peer applications call GSSAPI routines to delete the security context.

Routines described in this chapter are implemented in the Generic Security Service library (GSS$RTL.EXE for 64-bit interfaces, or GSS$RTL32.EXE for 32-bit interfaces) in SYS$LIBRARY.
gss_accept_sec_context — Establish a security context

C Prototype

```c
OM_uint32 gss_accept_sec_context(
    OM_uint32   * minor_status,
    gss_ctx_id_t * context_handle,
    gss_cred_id_t acceptor_cred_handle,
    gss_buffer_t input_token_buffer,
    gss_channel_bindings_t input_chan_bindings,
    gss_name_t * src_name,
    gss_OID   * mech_type,
    gss_buffer_t output_token,
    OM_uint32   * ret_flags,
    OM_uint32   * time_rec,
    gss_cred_id_t * delegated_cred_handle );
```

Arguments

- **minor_status (output)**: Mechanism-specific status code.
- **context_handle (input/output)**: The context handle for the new context. Supply GSS_C_NO_CONTEXT for the first call; use the value returned in subsequent calls. Once `gss_accept_sec_context` has returned a value via this argument, resources have been assigned to the corresponding context, and must be freed by the application after use with a call to `gss_delete_sec_context`.
- **acceptor_cred_handle (input)**: The credential handle claimed by the context acceptor. Specify GSS_C_NO_CREDENTIAL to accept the context as a default principal. If GSS_S_NO_CREDENTIAL is specified, but no default acceptor principal is defined, GSS_S_NO_CRED will be returned.
- **input_token_buffer (input)**: The token obtained from the remote application.
- **input_chan_bindings (input)**: Application-specified bindings. Allows the application to securely bind channel identification information to the security context. If channel bindings are not used, specify GSS_C_NO_CHANNEL_BINDINGS.
- **src_name (output)**: The authenticated name of the context initiator. After use, this name should be deallocated by passing it to `gss_release_name`. If not required, specify NULL.
- **mech_type (output)**: The security mechanism used. The returned OID value will be a pointer into static storage, and should be treated as read only by the caller (in particular, it does not need to be freed). If not required, specify NULL.
- **output_token (output)**: The token to be passed to the peer application. If the length field of the returned token buffer is zero, then no token need be passed to the peer application. If a nonzero length field is returned, the associated storage must be freed after use by the application with a call to `gss_release_buffer`. 
ret_flags (output)

A bit mask which contains various independent flags, each of which indicates that the context supports a specific service option. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically ANDed with the ret_flags value to test whether a given option is supported by the context. The flags are:

**GSS_C_DELEG_FLAG**

TRUE — Delegated credentials are available via the delegated_cred_handle argument.
FALSE — No credentials were delegated.

**GSS_C_MUTUAL_FLAG**

TRUE — The remote peer asked for mutual authentication.
FALSE — The remote peer did not ask for mutual authentication.

**GSS_C_REPLAY_FLAG**

TRUE — Replay of protected messages will be detected.
FALSE — Replayed messages will not be detected.

**GSS_C_SEQUENCE_FLAG**

TRUE — Out-of-sequence protected messages will be detected.
FALSE — Out-of-sequence messages will not be detected.

**GSS_C_CONF_FLAG**

TRUE — Confidentiality service may be invoked by calling the gss_wrap routine.
FALSE — No confidentiality service (via gss_wrap) is available. The gss_wrap routine will provide message encapsulation, data-origination authentication and integrity services only.

**GSS_C_INTEG_FLAG**

TRUE — Integrity service may be invoked by calling either the gss_get_mic or gss_wrap routine.
FALSE — Per-message integrity service is unavailable.

**GSS_C_ANON_FLAG**

TRUE — The initiator does not wish to be authenticated; the src_name argument (if requested) contains an anonymous internal name.
FALSE — The initiator has been authenticated normally.

**GSS_C_PROT_READY_FLAG**

TRUE — Protection services (as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG) are available if the accompanying status return value is either GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED.
FALSE — Protection services (as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG) are available only if the accompanying status return value is GSS_S_COMPLETE.

**GSS_C_TRANS_FLAG**
**GSSAPI (Generic Security Services Application Programming Interface)**

**gss_accept_sec_context — Establish a security context**

TRUE — The resultant security context may be transferred to other processes via a call to `gss_export_sec_context`.

FALSE — The security context is not transferable.

All other bits should be zero.

**time_rec (output)**

The number of seconds for which the context will remain valid. Specify NULL if not required.

**delegated_cred_handle (output)**

The credential handle for credentials received from the context initiator. Only valid if `deleg_flag` in `ret_flags` is TRUE, in which case an explicit credential handle (that is, not `GSS_C_NO_CREDENTIAL`) will be returned; if `deleg_flag` is false, `gss_accept_context` will set this argument to `GSS_C_NO_CREDENTIAL`. If a credential handle is returned, the associated resources must be released by the application after use with a call to `gss_release_cred`. Specify NULL if not required.

**Description**

This routine allows a remotely initiated security context between the application and a remote peer to be established. The routine may return an `output_token` that should be transferred to the peer application, where the peer application will present it to `gss_init_sec_context`. If no token need be sent, `gss_accept_sec_context` will indicate this by setting the length field of the `output_token` argument to zero. To complete the context establishment, one or more reply tokens may be required from the peer application; if so, `gss_accept_sec_context` will return a status flag of `GSS_S_CONTINUE_NEEDED`, in which case it should be called again when the reply token is received from the peer application, passing the token to `gss_accept_sec_context` via the `input_token` arguments.

Portable applications should be constructed to use the token length and return status to determine whether a token needs to be sent or waited for. A typical portable caller should always invoke `gss_accept_sec_context` within a loop. For example:

```c

GSS_Ctxt_id_t context_hdl = GSS_C_NO_CONTEXT;

do { receive_token_from_peer(input_token);
    maj_stat = gss_accept_sec_context( &min_stat,
                                &context_hdl,
                                cred_hdl,
                                input_token,
                                input_bindings,
                                &client_name,
                                &mech_type,
                                output_token,
                                &ret_flags,
                                &time_rec,
                                &deleg_cred);

    if (GSS_ERROR(maj_stat)) {
        report_error(maj_stat, min_stat);
    }
    if (output_token->length != 0) {
        send_token_to_peer(output_token);
        gss_release_buffer(&min_stat, output_token);
    }
    if (GSS_ERROR(maj_stat)) {
        if (context_hdl != GSS_C_NO_CONTEXT)
            gss_delete_sec_context( &min_stat,
                                    &context_hdl,
```
Whenever the routine returns a status that includes the value GSS_S_CONTINUE_NEEDED, the context is not fully established and the following restrictions apply to the output arguments:

- The value returned via the time_rec argument is undefined unless the accompanying ret_flags argument contains the bit GSS_C_PROT_READY_FLAG, indicating that per-message services may be applied in advance of a successful completion status. The value returned via the mech_type argument may be undefined until the routine returns a status of GSS_S_COMPLETE.
- The value of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG, and GSS_C_ANON_FLAG bits returned via the ret_flags argument contain the values that the implementation expects would be valid if context establishment were to succeed.
- The values of the GSS_C_PROT_READY_FLAG and GSS_C_TRANS_FLAG bits within ret_flags indicate the actual state at the time gss_accept_sec_context returns, whether or not the context is fully established.

Although this requires that GSSAPI implementations set the GSS_C_PROT_READY_FLAG in the final ret_flags returned to a caller (that is, when accompanied by a GSS_S_COMPLETE status code), applications should not rely on this behavior as the flag was not defined in Version 1 of the GSSAPI. Instead, applications should be prepared to use per-message services after a successful context establishment, according to the GSS_C_INTEG_FLAG and GSS_C_CONF_FLAG values.

- All other bits within the ret_flags argument will be set to zero. While the routine returns GSS_S_CONTINUE_NEEDED, the values returned via the ret_flags argument indicate the services that the implementation expects to be available from the established context.
- During context establishment, the information status bits GSS_S_OLD_TOKEN and GSS_S_DUPLICATE_TOKEN indicate fatal errors, and GSSAPI mechanisms return them in association with a routine error of GSS_S_FAILURE. This requirement for pairing did not exist in Version 1 of the GSSAPI specification, so applications that wish to run over Version 1 implementations must special-case these codes.

**Return Values**

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**
  Successfull completion.

- **GSS_S_CONTINUE_NEEDED**
  The service completed successfully and synchronously (returned only if the DDTM$M_SYNCH flag is set).

- **GSS_S_DEFECTIVE_TOKEN**
  Indicates that consistency checks performed on the input_token failed.

- **GSS_S_DEFECTIVE_CREDENTIAL**
  The options flags were invalid or the TID argument was omitted and the bid argument was not 0.

- **GSS_S_NO_CRED**
  The supplied credentials were not valid for context acceptance, or the credential handle did not reference any credentials.

- **GSS_S_CREDENTIALS_EXPIRED**
  The referenced credentials have expired.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_BAD_BINDINGS</td>
<td>The input_token contains different channel bindings to those specified via the input_chan_bindings argument.</td>
</tr>
<tr>
<td>GSS_S_NO_CONTEXT</td>
<td>Indicates that the supplied context handle did not refer to a valid context.</td>
</tr>
<tr>
<td>GSS_S_BAD_SIG</td>
<td>The input_token contains an invalid MIC.</td>
</tr>
<tr>
<td>GSS_S_OLD_TOKEN</td>
<td>The input_token was too old. This is a fatal error during context establishment.</td>
</tr>
<tr>
<td>GSS_S_DUPLICATE_TOKEN</td>
<td>The input_token is valid, but is a duplicate of a token already processed. This is a fatal error during context establishment.</td>
</tr>
<tr>
<td>GSS_S_BAD_MECH</td>
<td>The received token specified a mechanism that is not supported by the implementation or the provided credential.</td>
</tr>
</tbody>
</table>
gss_acquire_cred — Acquire credential handle

C Prototype

```c
OM_uint32 gss_acquire_cred(
    OM_uint32 * minor_status,
    gss_name_t desired_name,
    OM_uint32 time_req,
    gss_OID_set desired_mechs,
    gss_cred_usage_t cred_usage,
    gss_cred_id_t * output_cred_handle,
    gss_OID_set * actual_mechs,
    OM_uint32 * time_rec);
```

Arguments

- **minor_status (output)**: The mechanism-specific status code.
- **desired_name (input)**: The name of the principal whose credential should be acquired.
- **time_req (input)**: The number of seconds that credentials should remain valid. Specify GSS_C_INDIFINITE to request that the credentials have the maximum permitted lifetime.
- **desired_mechs (input)**: The set of underlying security mechanisms that may be used. GSS_C_NULL_OID_SET may be used to obtain an implementation-specific default.
- **cred_usage (input)**: One of the following values:
  - **GSS_C_BOTH** — Credentials may be used either to initiate or accept security contexts.
  - **GSS_C_INITIATE** — Credentials will only be used to initiate security contexts.
  - **GSS_C_ACCEPT** — Credentials will only be used to accept security contexts.
- **output_cred_handle (output)**: The returned credential handle. Resources associated with this credential handle must be released by the application after use with a call to gss_release_cred.
- **actual_mechs (output)**: The set of mechanisms for which the credential is valid. Storage associated with the returned OID-set must be released by the application after use with a call to gss_release_oid_set. Specify NULL if not required.
- **time_rec (output)**: The actual number of seconds for which the returned credentials will remain valid. If the implementation does not support expiration of credentials, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required.
GSSAPI (Generic Security Services Application Programming Interface)
gss_acquire_cred — Acquire credential handle

Description

This routine allows an application to acquire a handle for a pre-existing credential by name. GSSAPI implementations must impose a local access-control policy on callers of this routine to prevent unauthorized callers from acquiring credentials to which they are not entitled. This routine is not intended to provide a "login to the network" function, as such a function would result in the creation of new credentials rather than merely acquiring a handle to existing credentials.

If desired_name is GSS_C_NO_NAME, the call is interpreted as a request for a credential handle that will invoke default behavior when passed to gss_init_sec_context (if cred_usage is GSS_C_INITIATE or GSS_C_BOTH) or gss_accept_sec_context (if cred_usage is GSS_C_ACCEPT or GSS_C_BOTH).

This routine is expected to be used primarily by context acceptors.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**
  - Successful completion.
- **GSS_S_BAD_MECH**
  - Unavailable mechanism requested.
- **GSS_S_BAD_NAMETYPE**
  - The type contained within the desired_name argument is not supported.
- **GSS_S_BAD_NAME**
  - The value supplied for the desired_name argument is ill formed.
- **GSS_S_NO_CRED**
  - The supplied credentials were not valid for context acceptance, or the credential handle did not reference any credentials.
- **GSS_S_CREDENTIALS_EXPIRED**
  - The referenced credentials have expired.
**gss_add_cred — Construct credentials incrementally**

**C Prototype**

```c
OM_uint32 gss_add_cred(
    OM_uint32               * minor_status,
    gss_cred_id_t             input_cred_handle,
    gss_name_t                desired_name,
    gss_OID                   desired_mech,
    gss_cred_usage_t          cred_usage,
    OM_uint32                 initiator_time_req,
    OM_uint32                 acceptor_time_req,
    gss_cred_id_t            * output_cred_handle,
    gss_OID_set              * actual_mechs,
    OM_uint32                * initiator_time_rec,
    OM_uint32                * acceptor_time_rec );
```

**Arguments**

- **minor_status (output)**
  An implementation-specific status code.

- **input_cred_handle (input)**
  The credential to which a credential-element will be added. If GSS_C_NO_CREDENTIAL is specified, the routine will compose the new credential based on default behavior. (See description). Note that, while the credential handle is not modified by `gss_add_cred`, the underlying credential will be modified if `output_cred_handle` is NULL.

- **desired_name (input)**
  The name of the principal whose credential should be acquired.

- **desired_mech (input)**
  The underlying security mechanism with which the credential may be used.

- **cred_usage (input)**
  How the credential may be used. Valid values are as follows:
  - **GSS_C_INITIATE** — Credential will only be used to initiate security contexts.
  - **GSS_C_ACCEPT** — Credential will only be used to accept security contexts.

- **initiator_time_req (input)**
  The number of seconds that the credential should remain valid for initiating security contexts. This argument is ignored if the composed credentials are of type GSS_C_ACCEPT. Specify GSS_C_INDEFINITE to request that the credentials have the maximum permitted initiator lifetime.

- **acceptor_time_req (input)**
  The number of seconds that the credential should remain valid for accepting security contexts. This argument is ignored if the composed credentials are of type GSS_C_INITIATE. Specify GSS_C_INDEFINITE to request that the credentials have the maximum permitted initiator lifetime.

- **output_cred_handle (output)**
  The returned credential handle, containing the new credential-element and all the credential-elements from `input_cred_handle`. If a valid pointer to a `gss_cred_id_t` is supplied for this argument, `gss_add_cred`
creates a new credential handle containing all credential-elements from
the input_cred_handle and the newly acquired credential-element; if
NULL is specified for this argument, the newly acquired
credential-element will be added to the credential identified by
input_cred_handle.

The resources associated with any credential handle returned via this
argument must be released by the application after use with a call to
gss_release_cred.

actual_mechs (output)  The complete set of mechanisms for which the new credential is valid.
Storage for the returned OID-set must be freed by the application after
use with a call to gss_release_oid_set. Specify NULL if not required.

initiator_time_rec (output)  The actual number of seconds for which the returned credentials will
remain valid for initiating contexts using the specified mechanism. If the
implementation or mechanism does not support expiration of credentials,
the value GSS_C_INDEFINITE will be returned. Specify NULL if not
required.

acceptor_time_rec (output)  The actual number of seconds for which the returned credentials will
remain valid for accepting security contexts using the specified
mechanism. If the implementation or mechanism does not support
expiration of credentials, the value GSS_C_INDEFINITE will be
returned. Specify NULL if not required.

Description

This routine adds a credential-element to a credential. The credential-element is identified by the name of
the principal to which it refers. This routine is not intended to provide a "login to the network" function, as
such a function would involve the creation of new mechanism-specific authentication data, rather than
merely acquiring a GSSAPI handle to existing data.

If desired_name is GSS_C_NO_NAME, the call is interpreted as a request to add a credential element that
will invoke default behavior when passed to gss_init_sec_context (if cred_usage is GSS_C_INITIATE or
GSS_C_BOTH) or gss_accept_sec_context (if cred_usage is GSS_C_ACCEPT or GSS_C_BOTH).

This routine is expected to be used primarily by context acceptors, since implementations are likely to provide
mechanism-specific ways of obtaining GSSAPI initiator credentials from the system login process. Some
implementations may therefore not support the acquisition of GSS_C_INITIATE or GSS_C_BOTH
credentials via gss_acquire_cred for any name other than GSS_C_NO_NAME, or a name produced by
applying either gss_inquire_cred to a valid credential, or gss_inquire_context to an active context.

This routine can be used to either compose a new credential containing all credential-elements of the original
in addition to the newly acquired credential element, or to add the new credential-element to an existing
credential. If NULL is specified for the output_cred_handle argument, the new credential-element will be
added to the credential identified by input_cred_handle; if a valid pointer is specified for the
output_cred_handle argument, a new credential handle will be created.

If GSS_C_NO_CREDENTIAL is specified as the input_cred_handle, gss_add_cred will compose a
credential (and set the output_cred_handle argument accordingly) based on default behavior. That is, the
call will have the same effect as if the application had first made a call to gss_acquire_cred, specifying the
same usage and passing GSS_C_NO_NAME as the desired_name argument to obtain an explicit credential
handle embodying default behavior, passed this credential handle to gss_add_cred, and finally called
gss_release_cred on the first credential handle.
If GSS_C_NO_CREDENTIAL is specified as the input_cred_handle argument, a nonNULL output_cred_handle must be supplied.

**Return Values**

This routine returns one of the following GSS status codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_COMPLETE</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>GSS_S_BAD_MECH</td>
<td>Unavailable mechanism requested.</td>
</tr>
<tr>
<td>GSS_S_BAD_NAMETYPE</td>
<td>The type contained within the desired_name argument is not supported.</td>
</tr>
<tr>
<td>GSS_S_BAD_NAME</td>
<td>The value supplied for the desired_name argument is ill formed.</td>
</tr>
<tr>
<td>GSS_S_DUPLICATE_ELEMENT</td>
<td>The credential already contains an element for the requested mechanism with overlapping usage and validity period.</td>
</tr>
<tr>
<td>GSS_S_CREDENTIALS_EXPIRED</td>
<td>The required credentials could not be added because they have expired.</td>
</tr>
<tr>
<td>GSS_S_NO_CRED</td>
<td>No credentials were found for the specified name.</td>
</tr>
</tbody>
</table>
gss_add_oid_set_member — Add an object identifier to a set

C Prototype

```c
OM_uint32 gss_add_oid_set_member(
    OM_uint32 * minor_status,
    gss_OID member_oid,
    gss_OID_set * oid_set );
```

Arguments

- **minor_status** (output) An implementation-specific status code.
- **member_oid** (input) The object identifier to be copied into the set.
- **oid_set** (input/output) The set in which the object identifier should be inserted.

Description

This routine adds an object identifier to an object identifier set. It is intended for use in conjunction with `gss_create_empty_oid_set` when constructing a set of mechanism OIDs for input to `gss_acquire_cred`. The `oid_set` argument must refer to an OID-set that was created by GSSAPI (for example, a set returned by `gss_create_empty_oid_set`). GSSAPI creates a copy of the `member_oid` and inserts this copy into the set, expanding the storage allocated to the OID-set's elements array if necessary. The routine may add the new member OID anywhere within the elements array; if the `member_oid` is already present, the `oid_set` remains unchanged.

Return Values

This routine returns the following GSS status code:

- **GSS_S_COMPLETE** Successful completion.
gss_compare_name — Allow application to compare two internal names

C Prototype

```c
OM_uint32 gss_compare_name(
    OM_uint32          * minor_status,
    gss_name_t           name1,
    gss_name_t           name2,
    int                * name_equal );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **name1 (input)**: Internal-form name 1.
- **name2 (input)**: Internal-form name 2.
- **name_equal (output)**: A Boolean value.
  - TRUE — Names refer to the same entity.
  - FALSE — Names refer to different entities (strictly, the names are not known to refer to the same identity).

Description

This routine allows an application to compare two internal-form names to determine whether they refer to the same entity. If either name presented to `gss_compare_name` denotes an anonymous principal, the routine will indicate that the two names do not refer to the same identity.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
- **GSS_S_BAD_NAMETYPE**: The type contained within either name1 or name2 was unrecognized, or the names were of incomparable types.
- **GSS_S_BAD_NAME**: One or both of name1 or name2 was ill formed.
gss_canonicalize_name — Convert internal name to internal mechanism name

C Prototype

```c
OM_uint32 gss_canonicalize_name(
    OM_uint32               * minor_status,
    const gss_name_t          input_name,
    const gss_OID             mech_type,
    gss_name_t              * output_name );
```

Arguments

- **minor_status** (output) An implementation-specific status code.
- **input_name** (input) The name for which a canonical form is desired.
- **mech_type** (input) The authentication mechanism for which the canonical form of the name is desired. The desired mechanism must be specified explicitly; no default is provided.
- **output_name** (output) The resultant canonical name. Storage associated with this name must be freed by the application after use by a call to `gss_release_name`.

Description

This routine generates a canonical mechanism name (MN) from an arbitrary internal name. The mechanism name is the name that would be returned to a context acceptor on successful authentication of a context where the initiator used the `input_name` in a successful call to `gss_acquire_cred`, specifying an OID set containing `mech_type` as its only member, followed by a call to `gss_init_sec_context`, specifying `mech_type` as the authentication mechanism.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE** Successful completion.
- **GSS_S_BAD_MECH** The identified mechanism is not supported.
- **GSS_S_BAD_NAMETYPE** The provided internal name contains no elements that could be processed by the specified mechanism.
- **GSS_S_BAD_NAME** The `input_name` argument was ill formed.
gss_context_time — Check how much longer context is valid

C Prototype

```c
OM_uint32 gss_context_time(
    OM_uint32 * minor_status,
    gss_ctx_id_t context_handle,
    OM_uint32 * time_rec );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **context_handle (input)**: Identifies the context to be interrogated.
- **time_rec (output)**: The number of seconds that the context will remain valid. If the context has already expired, zero will be returned.

Description

Determines the number of seconds for which the specified context will remain valid.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
- **GSS_S_CONTEXT_EXPIRED**: The context has already expired.
- **GSS_S_NO_CONTEXT**: The context_handle argument did not identify a valid context.
gss_create_empty_oid_set — Create a set containing no object identifiers

C Prototype

OM_uint32 gss_create_empty_oid_set(
    OM_uint32 * minor_status,
    gss_OID_set * oid_set );

Arguments

minor_status (output) An implementation-specific status code.
oid_set (output) The empty object identifier set. The routine will allocate the
                  gss_OID_set_desc object, which the application must free after use with
                  a call to gss_release_oid_set.

Description

This routine creates an object identifier set containing no object identifiers, to which members may be
subsequently added using the gss_add_oid_set_member routine. These routines are intended to be used to
construct sets of mechanism object identifiers, for input to gss_acquire_cred.

Return Values

This routine returns the following GSS status code:

    GSS_S_COMPLETE    Successful completion.
gss_delete_sec_context — Delete a security context

**C Prototype**

```c
OM_uint32 gss_delete_sec_context(
    OM_uint32               * minor_status,
    gss_ctx_id_t            * context_handle,
    gss_buffer_t             output_token );
```

**Arguments**

- **minor_status (output)**: A mechanism-specific status code.
- **context_handle (input/output)**: A context handle identifying the context to delete. After deleting the context, the GSSAPI will set this context handle to GSS_C_NO_CONTEXT.
- **output_token (output)**: A token to be sent to the remote application to instruct it to also delete the context. It is recommended that applications specify GSS_C_NO_BUFFER for this argument, requesting local deletion only. If a buffer argument is provided by the application, the mechanism will either return a token in it, or set the length field of this token to zero to indicate to the application that no token is to be sent to the peer.

**Description**

This routine deletes a security context. The `gss_delete_sec_context` routine deletes the local data structures associated with the specified security context, and may generate an output_token, which when passed to the peer `gss_process_context_token` will instruct it to do likewise. No further security services may be obtained using the context specified by context_handle.

The output_token argument is retained for compatibility with Version 1 of the GSSAPI. It is recommended that both peer applications invoke `gss_delete_sec_context` passing the value GSS_C_NO_BUFFER for the output_token argument, indicating that no token is required, and that `gss_delete_sec_context` should simply delete local context data structures.

**Return Values**

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
- **GSS_S_FAILURE**: Failure. See minor_status for more information.
- **GSS_S_NO_CONTEXT**: No valid context was supplied.
**gss_display_name — Provide textual representation of opaque internal name**

**C Prototype**

```c
OM_uint32 gss_display_name(
    OM_uint32               * minor_status,
    gss_name_t                input_name,
    gss_buffer_t              output_name_buffer,
    gss_OID                 * output_name_type );
```

**Arguments**

- **minor_status (output)**: An implementation-specific status code.
- **input_name (input)**: The name to be displayed.
- **output_name_buffer (output)**: A buffer to receive the textual name string. The application must free storage associated with this name after use with a call to `gss_release_buffer`.
- **output_name_type (output)**: The type of the returned name. The returned `gss_OID` will be a pointer into static storage, and should be treated as read-only by the caller. (In particular, the application should not attempt to free it). Specify NULL if not required.

**Description**

This routine allows an application to obtain a textual representation of an opaque internal-form name for display purposes. The syntax of a printable name is defined by the GSSAPI implementation.

If `input_name` denotes an anonymous principal, the routine will return the `gss_OID` value `GSS_C_NT_ANONYMOUS` as the `output_name_type`, and a textual name that is syntactically distinct from all valid supported printable names in the `output_name_buffer`.

**Return Values**

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
- **GSS_S_BAD_NAMETYPE**: The type of `input_name` was not recognized.
- **GSS_S_BAD_NAME**: The `input_name` was ill formed.
gss_display_status — Convert GSSAPI status code to text for user display

C Prototype

OM_uint32 gss_display_status(
    OM_uint32               * minor_status,
    OM_uint32                 status_value,
    int                       status_type
    gss_OID                   mech_type,
    OM_uint32               * message_context,
    gss_buffer_t              status_string );

Arguments

minor_status (output) An implementation-specific status code.
status_value (input) The status value to be converted.
status_type (input) One of the following values:
  GSS_C_GSS_CODE — The status_value is a GSS status code.
  GSS_C_MECH_CODE — The status_value is a mechanism status code.
mech_type (input) The underlying mechanism (used to interpret a minor_status value). Supply GSS_C_NO_OID to obtain the system default.
message_context (input/output) This argument should be initialized to zero by the caller on the first call. If further messages are contained in the status_value argument, message_context will be nonzero on return, and this value should be passed back to subsequent calls, along with the same status_value, status_type, and mech_type arguments.
status_string (output) The textual interpretation of the status_value. Storage associated with this argument must be freed by the application after use with a call to gss_release_buffer.

Description

This routine allows an application to obtain a textual representation of a GSSAPI status code, for display to the user or for logging purposes. Since some status values may indicate multiple conditions, applications may need to call gss_display_status multiple times, each call generating a single text string. The message_context argument is used to store state information about which error messages have already been extracted from a given status_value; message_context must be initialized to zero by the application prior to the first call, and gss_display_status will return a nonzero value in this argument if there are further messages to extract.

The message_context argument contains all state information required by gss_display_status in order to extract further messages from the status_value; even when a nonzero value is returned in this argument, the application is not required to call gss_display_status again unless subsequent messages are desired. The following code extracts all messages from a given status code and prints them to SYS$ERROR.
GSSAPI (Generic Security Services Application Programming Interface)
gss_display_status — Convert GSSAPI status code to text for user display

```c
OM_uint32 message_context;
OM_uint32 status_code;
OM_uint32 maj_status;
OM_uint32 min_status;
gss_buffer_desc status_string;

message_context = 0;

do {
    maj_status = gss_display_status(&min_status,
                                   status_code,
                                   GSS_C_GSS_CODE,
                                   GSS_C_NO_OID,
                                   &message_context,
                                   &status_string);

    fprintf(stderr, "%.s\n",
            (int)status_string.length,
            (char *)status_string.value);
    gss_release_buffer(&min_status, &status_string);
} while (message_context != 0);

Return Values

This routine returns one of the following GSS status codes:

<table>
<thead>
<tr>
<th>GSS_S_COMPLETE</th>
<th>Successful completion.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_BAD_MECH</td>
<td>Indicates that translation in accordance with an unsupported mechanism type was requested.</td>
</tr>
<tr>
<td>GSS_S_BAD_STATUS</td>
<td>The status_value was not recognized, or the status_type was neither GSS_C_GSS_CODE nor GSS_C_MECH_CODE.</td>
</tr>
</tbody>
</table>
```
gss_duplicate_name — Create a copy of an internal name

C Prototype

```c
OM_uint32 gss_duplicate_name(
    OM_uint32           * minor_status,
    const gss_name_t    input_name,
    gss_name_t          * dest_name );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **input_name (input)**: The internal name to be duplicated.
- **dest_name (output)**: The resultant copy of `input_name`. Storage associated with this name must be freed by the application after use by a call to `gss_release_name`.

Description

This routine creates a duplicate of the existing internal name `input_name`. The new `dest_name` will be independent of `input_name` (that is, `input_name` and `dest_name` must both be released, and the release of one will not affect the validity of the other).

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
- **GSS_S_BAD_NAME**: The `input_name` argument was ill formed.
gss_export_name — Convert an internal mechanism name to export form

C Prototype

OM_uint32 gss_export_name(
    * minor_status,
    const gss_na input_name,
    gss_buffer_t exported_name);

Arguments

minor_status (output)  An implementation-specific status code.
input_name (input)  The mechanism name to be exported.
exported_name (output)  The canonical contiguous string form of input_name. Storage associated with this string must be freed by the application after use by a call to gss_release_buffer.

Description

This routine produces a canonical contiguous string representation of a mechanism name (MN), suitable for direct comparison (for example, with memcmp) for use in authorization functions (for example, matching entries in an access-control list). The input_name argument must specify a valid MN (that is, an internal name generated by gss_accept_sec_context or by gss_canonicalize_name).

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**  Successful completion.
- **GSS_S_NAME_NOT_MN**  The provided internal name was not a mechanism name.
- **GSS_S_BAD_NAME**  The provided internal name was ill formed.
- **GSS_S_BAD_NAMETYPE**  The internal name was of a type not supported by the GSSAPI implementation.
gss_export_sec_context — Transfer a security context to another process

C Prototype

```c
OM_uint32 gss_export_sec_context(  
    OM_uint32              * minor_status,  
    gss_ctx_id_t           * context_handle,  
    gss_buffer_t             interprocess_token );
```

Arguments

- minor_status (output) An implementation-specific status code.
- context_handle (input/output) The context handle identifying the context to transfer.
- interprocess_token (output) The token to be transferred to the target process. Storage associated with this token must be freed by the application after use with a call to gss_release_buffer.

Description

This routine is provided to support the sharing of work between multiple processes. It will typically be used by the context acceptor, in an application where a single process receives incoming connection requests and accepts security contexts over them, then passes the established context to one or more other processes for message exchange. The gss_export_sec_context routine deactivates the security context for the calling process and creates an interprocess token which, when passed to gss_import_sec_context in another process, will re-activate the context in the second process. Only a single instantiation of a given context may be active at any one time; a subsequent attempt by a context exporter to access the exported security context will fail.

The implementation may constrain the set of processes by which the interprocess token may be imported, either as a function of local security policy, or as a result of implementation decisions. For example, some implementations may constrain contexts to be passed only between processes that run under the same account, or which are part of the same process group.

The interprocess token may contain security-sensitive information (for example, cryptographic keys).

If the creation of the interprocess token is successful, all process-wide resources associated with the security context will be deallocated, and the context_handle will be set to GSS_C_NO_CONTEXT.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE** Successful completion.
- **GSS_S_CONTEXT_EXPIRED** The context has expired.
- **GSS_S_NO_CONTEXT** The context was invalid.
- **GSS_S_UNAVAILABLE** The operation is not supported.
gss_get_mic — Generate a cryptographic MIC for a message

C Prototype

```c
OM_uint32 gss_get_mic(
    OM_uint32               * minor_status,
    gss_ctx_id_t              context_handle,
    gss_qop_t                 qop_req,
    gss_buffer_t              message_buffer,
    gss_buffer_t              message_token );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **context_handle (input)**: Identifies the context on which the message will be sent.
- **qop_req (input)**: Specifies the requested quality of protection. Callers are encouraged, on portability grounds, to accept the default quality of protection offered by the chosen mechanism, which may be requested by specifying `GSS_C_QOP_DEFAULT` for this argument. If an unsupported protection strength is requested, `gss_get_mic` will return a status of `GSS_S_BAD_QOP`.
- **message_buffer (input)**: The message to be protected.
- **message_token (output)**: A buffer to receive the token. The application must free storage associated with this buffer after use with a call to `gss_release_buffer`.

Description

This routine supports data origin authentication and data integrity services. When `gss_get_mic` is invoked on an input message, it generates a cryptographic MIC, and places the MIC in a per-message token containing data items that allow underlying mechanisms to provide the specified security services. The original message, along with the generated per-message token, is passed to the remote peer; these two data elements are processed by `gss_verify_mic`, which validates the message in conjunction with the separate token. The `qop_req` argument allows a choice between several cryptographic algorithms.

This routine is functionally equivalent to the `gss_sign` routine. New code should use `gss_get_mic` instead of `gss_sign`. Although both routines are supported, `gss_sign` has been deprecated in the GSSAPI Version 2 specification.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Indicates that an integrity check, suitable for an established security context, was successfully applied and that the message and corresponding `per_msg_token` are ready for transmission.
- **GSS_S_CONTEXT_EXPIRED**: Indicates that context-related data items have expired, so that the requested operation cannot be performed.
<table>
<thead>
<tr>
<th>GSS_S_NO_CONTEXT</th>
<th>Indicates that the context_handle argument did not identify a valid context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_BAD_QOP</td>
<td>Indicates that the provided QOP value is not recognized or supported for the context.</td>
</tr>
</tbody>
</table>
gss_import_name — Convert a printable string to an internal form

C Prototype

OM_uint32 gss_import_name(
    OM_uint32               * minor_status,
    gss_buffer_t              input_name_buffer,
    gss_OID                   input_name_type,
    gss_name_t              * output_name );

Arguments

minor_status (output)  An implementation-specific status code.
input_name_buffer (input)  A buffer containing the contiguous string name to convert.
input_name_type (input)  The object ID specifying the type of printable name. Applications may specify either GSS_C_NO_OID to use a local system-specific printable syntax, or an OID recognized by the GSSAPI implementation to name a specific namespace.
output_name (output)  The returned name in internal form. Storage associated with this name must be freed by the application after use with a call to gss_release_name.

Description

This routine converts a contiguous string name to internal form. In general, the internal name returned (via the output_name argument) will not be an internal mechanism name; the exception to this is if the input_name_type indicates that the contiguous string provided via the input_name_buffer argument is of type GSS_C_NT_EXPORT_NAME, in which case the returned internal name will be a mechanism name for the mechanism that exported the name.

Return Values

This routine returns one of the following GSS status codes:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_COMPLETE</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>GSS_S_BAD_NAMETYPE</td>
<td>The input_name_type was unrecognized.</td>
</tr>
<tr>
<td>GSS_S_BAD_NAME</td>
<td>The input_name_buffer argument could not be interpreted as a name of the specified type.</td>
</tr>
<tr>
<td>GSS_S_BAD_MECH</td>
<td>The input name type was GSS_C_NT_EXPORT_NAME, but the mechanism contained within the input name is not supported.</td>
</tr>
</tbody>
</table>
gss_import_sec_context — Import a transferred context

C Prototype

```c
OM_uint32 gss_import_sec_context(
   OM_uint32               * minor_status,
   gss_buffer_t              interprocess_token,
   gss_ctx_id_t            * context_handle );
```

Arguments

- **minor_status (output)** An implementation-specific status code.
- **interprocess_token (input/output)** The token received from the exporting process.
- **context_handle (output)** The context handle of the newly reactivated context. Resources associated with this context handle must be released by the application after use with a call to `gss_delete_sec_context`.

Description

This routine allows a process to import a security context established by another process. A given interprocess token may be imported only once. See `gss_export_sec_context` for additional information.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE** Successful completion.
- **GSS_S_NO_CONTEXT** The token did not contain a valid context reference.
- **GSS_S_DEFECTIVE_TOKEN** The token was invalid.
- **GSS_S_UNAVAILABLE** The operation is unavailable.
- **GSS_S_UNAUTHORIZED** Local policy prevents the import of this context by the current process.
gss_indicate_mechs — Allow an application to determine which security mechanisms are available

C Prototype

```c
OM_uint32 gss_indicate_mechs(
    OM_uint32            * minor_status,
    gss_OID_set          * mech_set );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **mech_set (output)**: A set of implementation-supported mechanisms. The returned `gss_OID_set` value will be a dynamically allocated OID set that should be released by the caller after use with a call to `gss_release_oid_set`.

Description

This routine allows an application to determine which underlying security mechanisms are available.

Return Values

This routine returns the following GSS status code:

```
GSS_S_COMPLETE    Successful completion.
```
gss_init_sec_context — Establish a security context

C Prototype

```c
OM_uint32 gss_init_sec_context(
    OM_uint32               * minor_status,
    gss_cred_id_t             claimant_cred_handle,
    gss_ctx_id_t            * context_handle,
    gss_name_t                target_name,
    gss_OID                   mech_type,
    OM_uint32                 req_flags,
    OM_uint32                 time_req,
    gss_channel_bindings_t    input_chan_bindings,
    gss_buffer_t              input_token,
    gss_OID                 * actual_mech_type,
    gss_buffer_t              output_token,
    OM_uint32               * ret_flags,
    OM_uint32               * time_rec );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **claimant_cred_handle (input)**: A handle for credentials claimed. Supply GSS_C_NO_CREDENTIAL to act as a default initiator principal. If no default initiator is defined, the routine will return GSS_S_NO_CRED.
- **context_handle (input/output)**: The context handle for the new context. Supply GSS_C_NO_CONTEXT for the first call; use the value returned by the first call in continuation calls. Resources associated with this context handle must be released by the application after use with a call to gss_delete_sec_context.
- **target_name (input)**: The name of the target.
- **mech_type (input)**: The object ID of the desired mechanism. Supply GSS_C_NO_OID to obtain a mechanism-specific default.
- **req_flags (input)**: Contains various independent flags, each of which requests that the context support a specific service option. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically ORed together to form the bit-mask value. Valid values are:
  - **GSS_C_DELEG_FLAG**: TRUE — Delegate credentials to the remote peer. FALSE — Do not delegate.
  - **GSS_C_MUTUAL_FLAG**: TRUE — Request that the remote peer authenticate itself. FALSE — Authenticate self to the remote peer only.
  - **GSS_C_REPLAY_FLAG**:
GSSAPI (Generic Security Services Application Programming Interface)
gss_init_sec_context — Establish a security context

TRUE — Enable replay detection for messages protected with gss_wrap or gss_get_mic.
FALSE — Do not attempt to detect replayed messages.

GSS_C_SEQUENCE_FLAG
TRUE — Enable detection of out-of-sequence protected messages.
FALSE — Do not attempt to detect out-of-sequence messages.

GSS_C_CONF_FLAG
TRUE — Request that confidentiality service be made available (by calling gss_wrap).
FALSE — No per-message confidentiality service is required.

GSS_C_INTEG_FLAG
TRUE — Request that integrity service be made available (by calling either gss_get_mic or gss_wrap).
FALSE — No per-message integrity service is required.

GSS_C_ANON_FLAG
TRUE — Do not reveal the initiator’s identity to the acceptor.
FALSE — Authenticate normally.

time_req (input) The desired number of seconds for which the context should remain valid. Supply zero to request a default validity period.

input_chan_bindings (input) Application-specified bindings. Allows the application to securely bind channel identification information to the security context. Specify GSS_C_NO_CHANNEL_BINDINGS if channel bindings are not used.

input_token (input) The token received from the peer application. Supply GSS_C_NO_BUFFER, or a pointer to a buffer containing the value GSS_C_EMPTY_BUFFER on the initial call.

actual_mech_type (output) The actual mechanism used. The OID returned via this argument will be a pointer to static storage that should be treated as read only; in particular the application should not attempt to free it. Specify NULL if not required.

output_token (output) The token to be sent to the peer application. If the length field of the returned buffer is zero, no token need be sent to the peer application. Storage associated with this buffer must be freed by the application after use with a call to gss_release_buffer.

ret_flags (output) Contains various independent flags, each of which indicates that the context supports a specific service option. Specify NULL if not required. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically ANDed with the ret_flags value to test whether a given option is supported by the context. The flags are:

GSS_C_DELEG_FLAG
TRUE — Credentials were delegated to the remote peer.
FALSE — No credentials were delegated.
GSSAPI (Generic Security Services Application Programming Interface)

\[ \text{gss\_init\_sec\_context} \] — Establish a security context

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_C_MUTUAL_FLAG</td>
<td>TRUE — The remote peer has authenticated itself.</td>
</tr>
<tr>
<td></td>
<td>FALSE — The remote peer has not authenticated itself.</td>
</tr>
<tr>
<td>GSS_C_REPLAY_FLAG</td>
<td>TRUE — Replay of protected messages will be detected.</td>
</tr>
<tr>
<td></td>
<td>FALSE — Replayed messages will not be detected.</td>
</tr>
<tr>
<td>GSS_C_SEQUENCE_FLAG</td>
<td>TRUE — Out-of-sequence protected messages will be detected.</td>
</tr>
<tr>
<td></td>
<td>FALSE — Out-of-sequence messages will not be detected.</td>
</tr>
<tr>
<td>GSS_C_CONF_FLAG</td>
<td>TRUE — Confidentiality service may be invoked by calling the \text{gss_wrap} routine.</td>
</tr>
<tr>
<td></td>
<td>FALSE — No confidentiality service (via \text{gss_wrap}) is available. The \text{gss_wrap} routine will provide message encapsulation, data-origin authentication, and integrity services only.</td>
</tr>
<tr>
<td>GSS_C_INTEG_FLAG</td>
<td>TRUE — Integrity service may be invoked by calling either \text{gss_get_mic} or \text{gss_wrap} routines.</td>
</tr>
<tr>
<td></td>
<td>FALSE — Per-message integrity service is unavailable.</td>
</tr>
<tr>
<td>GSS_C_ANON_FLAG</td>
<td>TRUE — The initiator's identity has not been revealed, and will not be revealed if any emitted token is passed to the acceptor.</td>
</tr>
<tr>
<td></td>
<td>FALSE — The initiator's identity has been or will be authenticated normally.</td>
</tr>
<tr>
<td>GSS_C_PROT_READY_FLAG</td>
<td>TRUE — Protection services (as specified by the states of the \text{GSS_C_CONF_FLAG} and \text{GSS_C_INTEG_FLAG}) are available for use if the accompanying status return value is either \text{GSS_S_COMPLETE} or \text{GSS_S_CONTINUE_NEEDED}.</td>
</tr>
<tr>
<td></td>
<td>FALSE — Protection services (as specified by the states of the \text{GSS_C_CONF_FLAG} and \text{GSS_C_INTEG_FLAG}) are available only if the accompanying major status return value is \text{GSS_S_COMPLETE}.</td>
</tr>
<tr>
<td>GSS_S_TRANS_FLAG</td>
<td>TRUE — The resultant security context may be transferred to other processes via a call to \text{gss_export_sec_context}.</td>
</tr>
<tr>
<td></td>
<td>FALSE — The security context is not transferable.</td>
</tr>
</tbody>
</table>

All other bits should be set to zero.

| time_rec (output)          | The number of seconds for which the context will remain valid. If the implementation does not support credential expiration, the value \text{GSS_C_INDEFINITE} will be returned. Specify NULL if not required. |
**Description**

This routine indicates the establishment of a security context between the application and a remote peer. Initially, the `input_token` argument should be specified either as `GSS_C_NO_BUFFER`, or as a pointer to a `gss_buffer_desc` object whose length field contains the value zero. The routine may return an `output_token` that should be transferred to the peer application, where the peer application will present it to `gss_accept_sec_context`. If no token need be sent, `gss_init_sec_context` will indicate this by setting the length field of the `output_token` argument to zero. To complete the context establishment, one or more reply tokens may be required from the peer application; if so, `gss_init_sec_context` will return a status containing the supplementary information bit `GSS_S_CONTINUE_NEEDED`. In this case, `gss_init_sec_context` should be called again when the reply token is received from the peer application, passing the token to `gss_init_sec_context` via the `input_token` arguments.

Portable applications should be constructed to use the token length and return status to determine whether a token needs to be sent or waited for. Thus a typical portable caller should always invoke `gss_init_sec_context` within a loop:

```c
int context_established = 0;
gss_ctx_id_t context_hdl = GSS_C_NO_CONTEXT;
input_token->length = 0;

while (!context_established) {
    major_stat = gss_init_sec_context(&minor_stat,
        cred_hdl,              
        &context_hdl,         
        target_name,          
        &desired_mech,        
        &desired_services,    
        &desired_time,        
        input_bindings,       
        input_token,          
        &actual_mech,         
        output_token,         
        &actual_services,     
        &actual_time);

    if (GSS_ERROR(major_stat)) {
        report_error(major_stat, minor_stat);
    }

    if (output_token->length != 0) {
        send_token_to_peer(output_token);
        gss_release_buffer(&minor_stat, output_token);
    }

    if (GSS_ERROR(major_stat)) {
        if (context_hdl != GSS_C_NO_CONTEXT) 
            gss_delete_sec_context(&minor_stat, &context_hdl, 
                GSS_C_NO_BUFFER);
        break;
    }

    if (major_stat & GSS_S_CONTINUE_NEEDED) {
        receive_token_from_peer(input_token);
    } else {
        context_established = 1;
    }
```
Whenever the routine returns a status that indicates the value GSS_S_CONTINUE_NEEDED, the context is not fully established and the following restrictions apply to the output arguments:

- The value returned via the time_rec argument is undefined unless the accompanying ret_flags argument contains the bit GSS_C_PROT_READY_FLAG, indicating that per-message services may be applied in advance of a successful completion status, the value returned via the actual_mech_type argument is undefined until the routine returns a status value of GSS_S_COMPLETE.

- The values of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG and GSS_C_ANON_FLAG bits returned via the ret_flags argument contain the values that the implementation expects would be valid if context establishment were to succeed. In particular, if the application has requested a service such as delegation or anonymous authentication via the req_flags argument, and such a service is unavailable from the underlying mechanism, gss_init_sec_context generates a token that will not provide the service, and indicates via the ret_flags argument that the service will not be supported. The application may choose to abort the context establishment by calling gss_delete_sec_context (if it cannot continue in the absence of the service), or it may choose to transmit the token and continue context establishment (if the service was merely desired but not mandatory).

- The values of the GSS_C_PROT_READY_FLAG and GSS_C_TRANS_FLAG bits within ret_flags indicate the actual state at the time gss_init_sec_context returns, whether or not the context is fully established.

- GSSAPI implementations that support per-message protection are encouraged to set the GSS_C_PROT_READY_FLAG in the final ret_flags returned to a caller (that is, when accompanied by a GSS_S_COMPLETE status code). However, applications should not rely on this behavior, as the flag was not defined in Version 1 of the GSSAPI. Instead, applications should determine what per-message services are available after a successful context establishment according to the GSS_C_INTEG_FLAG and GSS_C_CONF_FLAG values.

If the initial call of gss_init_sec_context fails, a context object is not created, and the value of the context_handle argument is set to GSS_C_NO_CONTEXT to indicate this.

During context establishment, the informational status bits GSS_OLD_TOKEN and GSS_S_DUPLICATE_TOKEN indicate fatal errors, and GSSAPI mechanisms return them in association with a routine error of GSS_S_FAILURE. This requirement for pairing did not exist in Version 1 of the GSSAPI specification, so applications that wish to run over Version 1 implementations must special-case these codes.

**Return Values**

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**
  - Successful completion.

- **GSS_S_CONTINUE_NEEDED**
  - Indicates that a token from the peer application is required to complete the context and that gss_init_sec_context must be called again with that token.

- **GSS_S_DEFECTIVE_TOKEN**
  - Indicates that consistency checks performed on the input_token failed.
GSSAPI (Generic Security Services Application Programming Interface)
gss_init_sec_context — Establish a security context

GSS_S_DEFECTIVE_CREDENTIAL Indicates that consistency checks performed on the credential failed.

GSS_S_NO_CRED The supplied credentials were not valid for context initiation, or the credential handle did not reference any credentials.

GSS_S_CREDENTIALS_EXPIRED The referenced credentials have expired.

GSS_S_BAD_BINDINGS The input_token contains different channel bindings to those specified via the input_chan_bindings argument.

GSS_S_BAD_SIG The input_token contains an invalid MIC, or a MIC that could not be verified.

GSS_S_OLD_TOKEN The input_token was too old. This is a fatal error during context establishment.

GSS_S_DUPLICATE_TOKEN The input_token is valid, but is a duplicate of a token already processed. This is a fatal error during context establishment.

GSS_S_NO_CONTEXT Indicates that the supplied context handle did not refer to a valid context.

GSS_S_BAD_NAMETYPE The provided target_name argument contained an invalid or unsupported type of name.

GSS_S_BAD_NAME The provided target_name argument was ill formed.

GSS_S_BAD_MECH The specified mechanism is not supported by the provided credential, or is unrecognized by the implementation.
gss_inquire_context — Extract security context information

C Prototype

```c
OM_uint32 gss_inquire_context(
    OM_uint32               * minor_status,  
    gss_ctx_id_t              context_handle,  
    gss_name_t              * src_name,  
    gss_name_t              * targ_name,  
    OM_uint32               * lifetime_rec,  
    gss_OID                 * mech_type,  
    OM_uint32             * ctx_flags,  
    int                     * locally_initiated,  
    int                     * open );
```

Arguments

- **minor_status (output)**
  An implementation-specific status code.

- **context_handle (input)**
  A context handle identifying the context for which information is to be returned.

- **src_name (output)**
  The name of the context initiator. If the context was established using anonymous authentication, and if the application invoking `gss_inquire_context` is the context acceptor, an anonymous name will be returned. Storage associated with this name must be freed by the application after use with a call to `gss_release_name`.

- **targ_name (output)**
  The name of the context target. Storage associated with this name must be freed by the application after use with a call to `gss_release_name`. If the context acceptor did not authenticate itself, and if the initiator did not specify a target name in its call to `gss_init_sec_context`, the value `GSS_C_NO_NAME` will be returned. Specify NULL if not required.

- **lifetime_rec (output)**
  The number of seconds for which the context will remain valid. If the context has expired, this argument will be set to zero. If the implementation does not support credential expiration, the value `GSS_C_INDEFINITE` will be returned. Specify NULL if not required.

- **mech_type (output)**
  The security mechanism providing the context. The returned OID will be a pointer to static storage that should be treated as read only by the application; in particular the application should not attempt to free it. Specify NULL if not required.

- **ctx_flags (output)**
  Contains several independent flags, each of which indicates that the context supports (or is expected to support, if open is FALSE), a specific service option. If not needed, specify NULL. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically ANDed with the `ret_flags` value to test whether a given option is supported by the context. The flags are:

  **GSS_C_DELEG_FLAG**
  
  TRUE — Credentials were delegated from the initiator to the acceptor.
  
  FALSE — No credentials were delegated.
GSSAPI (Generic Security Services Application Programming Interface)
gss_inquire_context — Extract security context information

**GSS_C_MUTUAL_FLAG**
TRUE — The acceptor was authenticated to the initiator.
FALSE — The acceptor did not authenticate itself.

**GSS_C_REPLAY_FLAG**
TRUE — Replay of protected messages will be detected.
FALSE — Replay messages will not be detected.

**GSS_C_SEQUENCE_FLAG**
TRUE — Out-of-sequence protected messages will be detected.
FALSE — Out-of-sequence messages will not be detected.

**GSS_C_CONF_FLAG**
TRUE — Confidentiality service may be invoked by calling the `gss_wrap` routine.
FALSE — No confidentiality service (via `gss_wrap`) is available. The `gss_wrap` routine provides message encapsulation, data-origin authentication, and integrity services only.

**GSS_C_INTEG_FLAG**
TRUE — Integrity service may be invoked by calling either the `gss_get_mic` or `gss_wrap` routine.
FALSE — Per-message integrity service is unavailable.

**GSS_C_ANON_FLAG**
TRUE — The initiator's identity will not be revealed to the acceptor. The `src_name` argument (if requested) contains an anonymous internal name.
FALSE — The initiator has been authenticated normally.

**GSS_C_PROT_READY_FLAG**
TRUE — Protection services (as specified by the states of the `GSS_C_CONF_FLAG` and `GSS_C_INTEG_FLAG`) are available for use.
FALSE — Protection services (as specified by the states of the `GSS_C_CONF_FLAG` and `GSS_C_INTEG_FLAG`) are available only if the context is fully established (that is, if the open argument is nonzero).

**GSS_C_TRANS_FLAG**
TRUE — The resultant security context may be transferred to other processes via a call to `gss_export_sec_context`.
FALSE — The security context is not transferable.

**locally_initiated (output)**
A Boolean value. Specify NULL if not required.
TRUE if the caller is the context initiator.
FALSE if the caller is the acceptor.

**open (output)**
A Boolean value. Specify NULL if not required.
TRUE if the context is fully established.
FALSE if a context-establishment token is expected from the peer application.

Description

This routine is used to extract information describing characteristics of a security context. The caller must already have obtained a handle that refers to the context, although the context need not be fully established.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**
  Indicates that the referenced context is valid and that `ctx_flags`, `locally_initiated`, and `open` return values describe the corresponding characteristics of the context. If `open` is TRUE, `lifetime_rec` is also returned; if `open` is TRUE and the context peer's name is known, `src_name` and `targ_name` are valid in addition to the values listed previously. The `mech_type` value must be returned for contexts where `open` is TRUE and may be returned for contexts where `open` is FALSE.

- **GSS_S_NO_CONTEXT**
  Indicates that no valid context was recognized for the input `context_handle` provided. Return values other than `minor_status` are undefined.
gss_inquire_cred — Provide calling application with information about a credential

C Prototype

```c
OM_uint32 gss_inquire_cred(
    OM_uint32 * minor_status,
    gss_cred_id_t cred_handle,
    gss_name_t * name,
    OM_uint32 * lifetime,
    gss_cred_usage_t * cred_usage,
    gss_OID_set * mechanisms );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **cred_handle (input)**: A handle that refers to the target credential. Specify GSS_C_NO_CREDENTIAL to inquire about the default initiator principal.
- **name (output)**: The name whose identity the credential asserts. Storage associated with this name should be freed by the application after use with a call to gss_release_name. Specify NULL if not required.
- **lifetime (output)**: The number of seconds for which the credential will remain valid. If the credential has expired, this argument will be set to zero. If the implementation does not support credential expiration, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required.
- **cred_usage (output)**: How the credential may be used. Specify NULL if not required. Valid values are as follows:
  - GSS_C_INITIATE
  - GSS_C_ACCEPT
  - GSS_C_BOTH
- **mechanisms (output)**: The set of mechanisms supported by the credential. Storage associated with this OID set must be freed by the application after use with a call to gss_release_oid_set. Specify NULL if not required.

Description

This routine obtains information about a credential. The caller must already have obtained a handle that refers to the credential.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
Table of errors from `gss_inquire_credential`

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_NO_CRED</td>
<td>The referenced credentials could not be accessed.</td>
</tr>
<tr>
<td>GSS_S_DEFECTIVE_CREDENTIAL</td>
<td>The referenced credentials were invalid.</td>
</tr>
<tr>
<td>GSS_S_CREDENTIALS_EXPIRED</td>
<td>The referenced credentials have expired. If the lifetime argument was not passed as NULL, it will be set to zero.</td>
</tr>
</tbody>
</table>
gss_inquire_cred_by_mech — Obtain per-mechanism information about a credential

C Prototype

```c
OM_uint32 gss_inquire_cred_by_mech(
    OM_uint32               * minor_status,
    gss_cred_id_t             cred_handle,
    gss_OID                   mech_type,
    gss_name_t              * name,
    OM_uint32               * initiator_lifetime,
    OM_uint32             * acceptor_lifetime,
    gss_cred_usage_t        * cred_usage );
```

Arguments

- **minor_status (output)**
  A handle that refers to the target credential. Specify GSS_C_NO_CREDENTIAL to inquire about the default initiator principal.

- **mech_type (input)**
  The mechanism for which information should be returned.

- **name (output)**
  The name whose identity the credential asserts.

- **initiator_lifetime (output)**
  The number of seconds for which the credential will remain capable of initiating security contexts under the specified mechanism. If the credential can no longer be used to initiate contexts, or if the credential usage for this mechanism is GSS_C_ACCEPT, this argument will be set to zero. If the implementation does not support expiration of initiator credentials, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required.

- **acceptor_lifetime (output)**
  The number of seconds for which the credential will remain capable of accepting security contexts under the specified mechanism. If the credential can no longer be used to accept contexts, or if the credential usage for this mechanism is GSS_C_INITIATE, this argument will be set to zero. If the implementation does not support expiration of acceptor credentials, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required.

- **cred_usage (output)**
  How the credential may be used with the specified mechanism. Specify NULL if not required. Valid values are as follows:

  - **GSS_C_INITIATE**
  - **GSS_C_ACCEPT**
  - **GSS_C_BOTH**

Description

This routine obtains per-mechanism information about a credential.
Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
- **GSS_S_NO_CRED**: The referenced credentials could not be accessed.
- **GSS_S_DEFECTIVE_CREDENTIAL**: The referenced credentials were invalid.
gss_inquire_names_for_mech — Return set of supported nametypes

C Prototype

```c
OM_uint32 gss_inquire_names_for_mech(
    OM_uint32               * minor_status,
    gss_OID                   mechanism,
    gss_OID_set             * name_types );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **mechanism (input)**: The mechanism to be interrogated.
- **name_types (output)**: The set of name-types supported by the specified mechanism. The returned OID set must be freed by the application after use with a call to `gss_release_oid_set`.

Description

This routine returns the set of nametypes supported by the specified mechanism.

Return Values

This routine returns the following GSS status code:

```
GSS_S_COMPLETE           Successful completion.
```
gss_process_context_token — Pass a security context to the security service

C Prototype

```c
OM_uint32 gss_process_context_token(
    OM_uint32 * minor_status,
    gss_ctx_id_t context_handle,
    gss_buffer_t token_buffer);
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **context_handle (input)**: The context handle of the context on which the token is to be processed.
- **token_buffer (input)**: A pointer to the token to process.

Description

This routine provides a way to pass an asynchronous token to the security service. Most context-level tokens are emitted and processed synchronously by `gss_init_sec_context` and `gss_accept_sec_context`, and the application is informed as to whether further tokens are expected by the GSS_C_CONTINUE_NEEDED status return. Occasionally, a mechanism may need to emit a context-level token at a point when the peer entity is not expecting a token. For example, the initiator's final call to `gss_init_sec_context` may emit a token and return a status of GSS_S_COMPLETE, but the acceptor's call to `gss_accept_sec_context` may fail. The acceptor's mechanism may wish to send a token containing an error indication to the initiator, but the initiator is not expecting a token at this point, believing that the context is fully established. The `gss_process_context_token` routine provides a way to pass such a token to the mechanism at any time.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
- **GSS_S_DEFECTIVE_TOKEN**: Indicates that consistency checks performed on the token failed.
- **GSS_S_FAILURE**: Failure. See `minor_status` for more information.
- **GSS_S_NO_CONTEXT**: The `context_handle` did not refer to a valid context.
gss_release_buffer — Free storage associated with a buffer

C Prototype

```c
OM_uint32 gss_release_buffer(
    OM_uint32              * minor_status,
    gss_buffer_t             buffer );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **buffer (input/output)**: The storage associated with the buffer will be deleted. The `gssBufferDesc` object will not be freed, but its length field will be zeroed.

Description

This routine frees storage associated with a buffer. The storage must have been allocated by a GSSAPI routine. In addition to freeing the associated storage, the routine will zero the length field in the descriptor to which the `buffer` argument refers. Any `buffer` object returned by a GSSAPI routine may be passed to `gss_release_buffer` (even if there is no storage associated with the buffer).

Return Values

This routine returns the following GSS status code:

- **GSS_S_COMPLETE**: Successful completion.
gss_release_cred — Mark a credential for deletion

C Prototype

```c
OM_uint32 gss_release_cred(
    OM_uint32               * minor_status,
    gss_cred_id_t           * cred_handle );
```

Arguments

- **minor_status (output)**: A mechanism-specific status code.
- **cred_handle (input/output)**: A buffer containing an opaque credential handle identifying the credential to be released. If GSS_C_NO_CREDENTIAL is supplied, the routine will complete successfully, but will do nothing.

Description

This routine informs GSSAPI that the specified credential handle is no longer required by the application, and frees associated resources. When all processes have released a credential, it will be deleted.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Successful completion.
- **GSS_S_NO_CRED**: The credentials could not be accessed.
**gss_release_name — Free storage associated with an internal name that was allocated by a GSSAPI routine**

**C Prototype**

```c
OM_uint32 gss_release_name(
    OM_uint32               * minor_status,
    gss_name_t              * input_name );
```

**Arguments**

- minor_status (output) An implementation-specific status code.
- input_name (input/output) The name to be deleted.

**Description**

This routine frees GSSAPI allocated storage associated with an internal form name.

**Return Values**

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**  
  Successful completion.

- **GSS_S_BAD_NAME**  
  The `input_name` argument did not contain a valid name.
gss_release_oid_set — Free storage associated with a gss_OID_set object

C Prototype

OM_uint32 gss_release_oid_set(
    OM_uint32   * minor_status,
    gss_OID_set * set);

Arguments

minor_status (output) An implementation-specific status code.
set (input) The gss_OID_set whose storage is to be deleted.

Description

This routine frees storage associated with a GSSAPI generated gss_OID_set object. The set argument must refer to an OID-set that was returned from a GSSAPI routine. The gss_release_oid_set routine frees the storage associated with each individual member OID, the OID set's elements array, and the gss_OID_set_desc.

Return Values

This routine returns the following GSS status code:

GSS_S_COMPLETE Successful completion.
**gss_test_oid_set_member** — Determine whether an object identifier is a member of the set

### C Prototype

```c
OM_uint32 gss_test_oid_set_member(
    OM_uint32               * minor_status,
    gss_OID                   member,
    gss_OID_set               set,
    int                     * present );
```

### Arguments

- **minor_status (output)**: An implementation-specific status code.
- **member (input)**: The object identifier whose presence is to be tested.
- **set (input)**: The object identifier set.
- **present (output)**: A Boolean value:
  - TRUE — The specified OID is a member of the set.
  - FALSE — The specified OID is not a member of the set.

### Description

This routine interrogates an object identifier set to determine whether a specified object identifier is a member. It is intended to be used with OID sets returned by `gss_indicate_mechs`, `gss_acquire_cred`, and `gss_inquire_cred`, but will also work with user-generated sets.

### Return Values

This routine returns the following GSS status code:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_COMPLETE</td>
<td>Successful completion.</td>
</tr>
</tbody>
</table>
gss_unwrap — Verify a message with attached MIC and decrypt message content

C Prototype

```c
OM_uint32 gss_unwrap(
    OM_uint32               * minor_status,
    gss_ctx_id_t              context_handle,
    gss_buffer_t              input_message_buffer,
    gss_buffer_t              output_message_buffer,
    int                     * conf_state,
    gss_qop_t               * qop_state );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **context_handle (input)**: Identifies the context in which the message arrived.
- **input_message_buffer (input)**: The protected message.
- **output_message_buffer (output)**: A buffer to receive the unwrapped message. Storage associated with this buffer must be freed by the application after use with a call to `gss_release_buffer`.
- **conf_state (output)**: A Boolean value indicating which services have been applied. Specify NULL if not required.
  - TRUE — Confidentiality and integrity protection services have been applied.
  - FALSE — Only integrity service has been applied.
- **qop_state (output)**: The quality of protection provided. Specify NULL if not required.

Description

This routine converts a message previously protected by `gss_wrap` back to a usable form, verifying the embedded Message Integrity Code (MIC). The `conf_state` argument indicates whether the message was encrypted; the `qop_state` argument indicates the strength of the protection that was used to provide the confidentiality and integrity services.

This routine is functionally equivalent to the `gss_unseal` routine. New code should use `gss_unwrap` instead of `gss_unseal`. Although both routines are supported, `gss_unseal` has been deprecated in the GSSAPI Version 2 specification.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Indicates that the `input_message_buffer` was successfully processed and that the `output_message_buffer` is ready for transmission.
GSSAPI (Generic Security Services Application Programming Interface)
gss_unwrap — Verify a message with attached MIC and decrypt message content

- **GSS_S_DEFECTIVE_TOKEN**: Indicates that the input_message_buffer was successfully processed and that the output_message_buffer is ready for transmission.

- **GSS_S_BAD_SIG**: Indicates that consistency checks performed on the token extracted from the input_message_buffer failed, preventing further processing from being performed with that token.

- **GSS_S_DUPLICATE_TOKEN**: Indicates that the MIC extracted from the input_message_buffer contains an incorrect integrity check for the message.

- **GSS_S_OLD_TOKEN**: The token extracted from the input_message_buffer is valid, and contained a correct MIC for the message, but is a duplicate of a token already processed. This is a fatal error during context establishment.

- **GSS_S_UNSEQ_TOKE**: Indicates that the token was valid, and contained a correct MIC for the message, but has been verified out of sequence; a later token has already been received.

- **GSS_S_GAP_TOKEN**: Indicates that the token was valid, and contained a correct MIC for the message, but has been verified out of sequence; an earlier expected token has not yet been received.

- **GSS_S_CONTEXT_EXPIRED**: Indicates that context-related data items have expired, so that the requested operation cannot be performed.

- **GSS_S_NO_CONTEXT**: Indicates that no valid context was recognized for the input context_handle provided.
gss_verify_mic — Check that a cryptographic MIC fits the applied message

C Prototype

```c
OM_uint32 gss_verify_mic(
    OM_uint32               * minor_status,
    gss_ctx_id_t              context_handle,
    gss_buffer_t              message_buffer,
    gss_buffer_t              message_token,
    gss_qop_t               * qop_state );
```

Arguments

- `minor_status` (output): An implementation-specific status code.
- `context_handle` (input): Specifies the context on which the message arrived.
- `message_buffer` (input): Specifies the message to be verified.
- `message_token` (input): Specifies the token to be associated with the message.
- `qop_state` (output): Returns the quality of protection gained from the MIC. Specify NULL if not required.

Description

This routine checks that a cryptographic MIC, contained in the `message_token` argument, fits the message in the `message_buffer` argument. The `qop_state` argument allows a message recipient to determine the strength of protection that was applied to the message.

This routine is functionally equivalent to the `gss_verify` routine. New code should use `gss_verify_mic` instead of `gss_verify`. Although both routines are supported, `gss_verify` has been deprecated in the GSSAPI Version 2 specification.

Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**: Indicates that the message was successfully verified.
- **GSS_S_DEFECTIVE_TOKEN**: Indicates that consistency checks performed on the received `message_token` failed, preventing further processing from being performed with that token.
- **GSS_S_BAD_SIG**: Indicates that the received `message_token` contains an incorrect MIC for the message.
- **GSS_S_DUPLICATE_TOKEN**: The `message_token` was valid, and contained a correct MIC for the message, but is a duplicate of a token already processed. This is a fatal error during context establishment.
GSSAPI (Generic Security Services Application Programming Interface)

gss_verify_mic — Check that a cryptographic MIC fits the applied message

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_OLD_TOKEN</td>
<td>The message_token was valid, and contained a correct MIC for the message, but the message_token was too old to check for duplication. This is a fatal error during context establishment.</td>
</tr>
<tr>
<td>GSS_S_UNSEQ_TOKEN</td>
<td>Indicates that the cryptographic check value on the received message was correct, and the message_token contained a correct MIC, but the token has been verified out of sequence; a later token has already been received.</td>
</tr>
<tr>
<td>GSS_S_GAP_TOKEN</td>
<td>Indicates that the cryptographic check value on the received message was correct, and the message_token contained a correct MIC, but the token has been verified out of sequence; an earlier expected token has not yet been received.</td>
</tr>
<tr>
<td>GSS_S_CONTEXT_EXPIRED</td>
<td>Indicates that context-related data items have expired, so that the requested operation cannot be performed.</td>
</tr>
<tr>
<td>GSS_S_NO_CONTEXT</td>
<td>Indicates that no valid context was recognized for the input context_handle provided.</td>
</tr>
</tbody>
</table>
gss_wrap — Attach a MIC to a message and encrypt the message

C Prototype

```c
OM_uint32 gss_wrap(
    OM_uint32               * minor_status,
    gss_ctx_id_t              context_handle,
    int                       conf_req_flag,
    gss_qop_t                 qop_req,
    gss_buffer_t              input_message_buffer,
    int                       * conf_state,
    gss_buffer_t              output_message_buffer );
```

Arguments

- **minor_status (output)**: An implementation-specific status code.
- **context_handle (input)**: Identifies the context on which the message will be sent.
- **conf_req_flag (input)**: A Boolean value indicating which services are to be used. TRUE — Both confidentiality and integrity services are requested. FALSE — Only integrity service is requested.
- **qop_req (input)**: Specifies the required quality of protection. A mechanism-specific default may be requested by setting `qop_req` to `GSS_C_QOP_DEFAULT`. If an unsupported protection strength is requested, `gss_wrap` will return a status of `GSS_S_BAD_QOP`.
- **input_message_buffer (input)**: The message to be protected.
- **conf_state (output)**: A Boolean value indicating which services have been applied. Specify NULL if not required. TRUE — Confidentiality, data origin authentication and integrity services have been applied. FALSE — Only integrity and data origin services have been applied.
- **output_message_buffer (output)**: The buffer to receive the protected message. Storage associated with this message must be freed by the application after use with a call to `gss_release_buffer`.

Description

This routine attaches a cryptographic MIC and optionally encrypts the specified `input_message_buffer`. The `output_message_buffer` contains both the MIC and the message. The `qop_req` argument allows a choice between several cryptographic algorithms.

This routine is functionally equivalent to the `gss_seal` routine. New code should use `gss_wrap` instead of `gss_seal`. Although both routines are supported, `gss_seal` has been deprecated in the GSSAPI Version 2 specification.
Return Values

This routine returns one of the following GSS status codes:

- **GSS_S_COMPLETE**
  Indicates that the input_message_buffer was successfully processed and that the output_message_buffer is ready for transmission.

- **GSS_S_CONTEXT_EXPIRED**
  Indicates that context-related data items have expired, so that the requested operation cannot be performed.

- **GSS_S_NO_CONTEXT**
  Indicates that the context_handle argument did not identify a valid context.

- **GSS_S_BAD_QOP**
  Indicates that the provided QOP value is not recognized or supported for the context.
gss_wrap_size_limit — Check expected size of wrapped output

C Prototype

```c
OM_uint32 gss_wrap_size_limit(
    OM_uint32               * minor_status,
    gss_ctx_id_t              context_handle,
    int                       conf_req_flag,
    gss_qop_t                 qop_req,
    OM_uint32                 req_output_size,
    OM_uint32               * max_input_size );
```

Arguments

- minor_status (output): An implementation-specific status code.
- context_handle (input): A handle that refers to the security over which the messages will be sent.
- conf_req_flag (input): A Boolean value indicating whether gss_wrap will be asked to apply confidentiality protection in addition to integrity protection.
  - TRUE — Both confidentiality and integrity services are requested.
  - FALSE — Only integrity service is requested.
- qop_req (input): Specifies the requested quality of protection that gss_wrap will be asked to provide. Callers are encouraged, on portability grounds, to accept the default quality of protection offered by the chosen mechanism, which may be requested by specifying GSS_C_QOP_DEFAULT for this argument.
- req_output_size (input): The desired maximum size for tokens emitted by gss_wrap.
- max_input_size (output): The maximum input message size that may be presented to gss_wrap in order to guarantee that the emitted token shall be no larger than req_output_size bytes.

Description

This routine allows an application to determine the maximum message size that, if presented to gss_wrap with the same conf_req_flag and qop_req arguments, will result in an output token containing no more than req_output_size bytes.

This call is intended for use by applications that communicate over protocols that impose a maximum message size. It enables the application to fragment messages prior to applying protection.

This call is intended for use by applications that communicate over protocols that impose a maximum message size. It enables the application to fragment messages prior to applying protection.

Successful completion of this call does not guarantee that gss_wrap will be able to protect a message of length max_input_size bytes, since this ability may depend on the availability of system resources at the time that gss_wrap is called.
GSSAPI (Generic Security Services Application Programming Interface)
gss_wrap_size_limit — Check expected size of wrapped output

Return Values

This routine returns one of the following GSS status codes:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS_S_COMPLETE</td>
<td>Indicates a successful token size determination: an input message with a length in octets equal to the returned max_input_size value will, when passed to gss_wrap for processing on the context identified by the context_handle argument with the confidentiality request state as provided in conf_req_flag and with the quality of protection specifier provided in the qop_req argument, yield an output token no larger than the value of the provided req_output_size argument.</td>
</tr>
<tr>
<td>GSS_S_CONTEXT_EXPIRED</td>
<td>Indicates that the provided input context_handle is recognized, but that the referenced context has expired. Return values other than minor_status are undefined.</td>
</tr>
<tr>
<td>GSS_S_NO_CONTEXT</td>
<td>Indicates that no valid context was recognized for the input context_handle provided. Return values other than minor_status are undefined.</td>
</tr>
<tr>
<td>GSS_S_BAD_QOP</td>
<td>Indicates that the provided QOP value is not recognized or supported for the context.</td>
</tr>
</tbody>
</table>
This chapter describes the C language bindings for the routines that make up the KRB5 Application Programming Interface.

The APIs in the following list are now obsolete, and their use should be avoided. (A future version of Kerberos may remove these APIs.) The column on the right indicates the API that should be used as a substitute for the obsolete API.

Table 6-1 Obsolete and Replacement APIs

<table>
<thead>
<tr>
<th>Obsolete API</th>
<th>Replacement API</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_auth_con_getlocalsubkey</td>
<td>krb5_auth_con_getsendsubkey</td>
</tr>
<tr>
<td>krb5_auth_con_getremotesubkey</td>
<td>krb5_auth_con_getrecvsubkey</td>
</tr>
<tr>
<td>krb5_auth_con_initivector</td>
<td>None</td>
</tr>
<tr>
<td>krb5_get_in_tkt_with_skey</td>
<td>None</td>
</tr>
<tr>
<td>krb5_get_in_tkt_with_password</td>
<td>krb5_get_init_creds_password</td>
</tr>
<tr>
<td>krb5_get_in_tkt_with_keytab</td>
<td>krb5_get_init_creds_keytab</td>
</tr>
<tr>
<td>krb5_get_in_tkt</td>
<td>None</td>
</tr>
</tbody>
</table>

NOTE Additional Kerberos KRB5 APIs are not documented in this manual. The APIs themselves are included in the Kerberos for OpenVMS library (KRBS$RTL.EXE for 64 bit interfaces, or KRBS$RTL32.EXE for 32 bit interfaces) in SYS$LIBRARY.
**krb5_425_conv_principal** — Convert a Kerberos V4 principal name to V5 format

**C Prototype**

```c
krb5_error_code krb5_425_conv_principal(
    krb5_context context,
    const char *name,
    const char *instance,
    const char *realm,
    krb5_principal *princ);
```

**Arguments**

- **context** (input/output): The context structure.
- **name** (input): Kerberos V4 name.
- **instance** (input): Kerberos V4 instance.
- **realm** (input): Kerberos V4 realm.
- **principal** (output): Kerberos V5 principal name.

**Description**

This routine builds a principal \texttt{princ} from a V4 specification made up of \texttt{name.instance@realm}. The routine is site customized to convert the V4 naming scheme to a V5 scheme. For instance, the V4 \texttt{rcmd} is changed to \texttt{host}.

The returned principal should be freed with \texttt{krb5_free_principal}.

**Return Values**

This routine returns the following KRB5 status code:

- 0: Successful completion.
**krb5_524_conv_principal — Separate a Kerberos V5 principal into components**

**C Prototype**

```c
krb5_error_code krb5_524_conv_principal (            
  krb5_context            context,            
  krb5_const_principal    princ,            
  char                    *name,            
  char                    *inst,            
  char                    *realm );
```

**Arguments**

- **context** (input/output) The context structure.
- **princ** (input) The Kerberos V5 principal.
- **name** (output) The principal name.
- **inst** (output) The principal instance name.
- **realm** (output) The principal realm name.

**Description**

This routine separates a Kerberos V5 principal into `name`, `instance`, and `realm`.

**Return Values**

This routine returns the following KRB5 status codes:

- 0 Successful completion.
- KRB5_INVALID_PRINCIPAL Invalid principal name.
- KRB5_CONFIG_CANTOPEN Can't open/find Kerberos configuration file.
krb5_524_convert_creds — Convert Kerberos V5 credentials to V4

C Prototype

```c
krb5_error_code krb5_524_convert_creds (  
    krb5_context                    context,  
    krb5_creds                      *v5creds,  
    CREDENTIALS                     *v4creds );
```

Arguments

- **context** (input/output): The context structure.
- **v5creds** (input): A pointer to the Kerberos V5 credentials to be converted.
- **v4creds** (output): A pointer to the Kerberos V4 credential structure to be filled in.

Description

This routine takes a set of Kerberos V5 credentials, and converts them to V4 format.

Return Values

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **-1**: Bad key.
- **KRB5_INVALID_PRINCIPAL**: Invalid principal name.
- **KRB5_CONFIG_CANTOPEN**: Can't open/find Kerberos configuration file.
- **KRB5_REALM_UNKNOWN**: Unknown realm.
- **KRB524_KRB5_DISABLED**: Kerberos V4 compatibility is disabled.
- **ENOMEM**: Insufficient memory.
krb5_address_compare — Compare two addresses

C Prototype

```c
krb5_boolean krb5_address_compare(
    krb5_context context,
    const krb5_address *addr1,
    const krb5_address *addr2);
```

Arguments

- context (input/output): The context structure.
- addr1 (input): The first address to compare.
- addr2 (input): The second address to compare.

Description

This routine compares two Kerberos addresses.

Return Values

This routine returns one of the following KRB5 status codes:

- TRUE: The two addresses are the same.
- FALSE: The two addresses are different.
The krb5_address_order function is used to compare two addresses and return an ordering on the two addresses.

**C Prototype**

```c
int krb5_address_order(
    krb5_context context,
    const krb5_address *addr1,
    const krb5_address *addr2);
```

**Arguments**

- `context` (input/output) The context structure.
- `addr1` (input) The first address to compare.
- `addr2` (input) The second address to compare.

**Description**

This routine returns an ordering on the two addresses.

**Return Values**

This routine returns one of the following KRB5 status codes:

- `= 0` The two addresses are the same.
- `< 0` First address is less than second.
- `> 0` First address is greater than second.
**C Prototype**

```c
krb5_boolean krb5_address_search(
    krb5_context               context,
    const krb5_address         *addr,
    krb5_address  * krb5_const  *addrlist );
```

**Arguments**

- `context (input/output)` The context structure.
- `addr (input)` The address to search for.
- `addrlist (input)` The address list to search, as an array of addresses. The last entry in the array must be a NULL pointer. Specify NULL for this argument if no address list is present.

**Description**

This routine searches `addrlist` for the address in `addr`.

**Return Values**

This routine returns one of the following KRB5 status codes:

- **TRUE** `addr` is listed in `addrlist`, or `addrlist` is NULL.
- **FALSE** `addr` is not listed in `addrlist`. 
krb5_aname_to_localname — Convert a principal name to a local name

C Prototype

krb5_error_code krb5_aname_to_localname(
    krb5_context context,
    krb5_const_principal aname,
    int lnsize,
    char *lname );

Arguments

context (input) The context structure.
aname (input) A principal name.
lnsize (input) Specifies the maximum length name that is to be filled into lname.
lname (output) The local name.

Description

This routine converts a principal name aname to a local name suitable for use by programs wishing a translation to an environment-specific name (for example, user account name).
The translation will be NULL terminated in all nonerror returns.

Return Values

This routine returns the following KRB5 status code:

    System errors.
krb5_appdefault_boolean — Check Boolean values in appdefault

C Prototype

```c
void krb5_appdefault_boolean(
    krb5_context    context,
    const char      *appname,
    const krb5_data *realm,
    const char      *option,
    int             default_value
int             *ret_value );
```

Arguments

context (input)  The context structure.

appname (input)  A pointer to the application name.

realm (input) A pointer to the Kerberos realm name.

option (input)  A pointer to the option to be checked.

default_value (input) A default Boolean value to return if no match is found.

ret_value (output) A pointer to the returned Boolean value.

Description

This routine checks the [appdefaults] section of the krb5.conf file. The ret_value argument returns the Boolean value of the particular option passed in the option argument. The appname argument provides the application name (for example, Telnet) whose option is being checked.

Use krb5_appdefault_string for checking string values in the [appdefaults] section of krb5.conf.

Return Values

None.
KRB5 (Kerberos V5) Application Programming Interface

**krb5_appdefault_string — Check string values in appdefault**

**C Prototype**

```c
void krb5_appdefault_string (
  krb5_context    context,
  const char      *appname,
  const krb5_data *realm,
  const char      *option,
  const char      *default_value,
  char            **ret_value );
```

**Arguments**

- **context (input)**: The context structure.
- **appname (input)**: A pointer to the application name.
- **realm (input)**: A pointer to the Kerberos realm name.
- **option (input)**: A pointer to the option to be checked.
- **default_value (input)**: A default Boolean value to return if no match is found.
- **ret_value (output)**: A pointer to the returned string value.

**Description**

This routine checks the `[appdefaults]` section of the `krb5.conf` file. The `ret_value` argument returns the string value of the particular option passed in the `option` argument. The `appname` argument provides the application name (for example, Telnet) whose option is being checked.

Use `krb5_appdefault_boolean` for checking Boolean values in the `[appdefaults]` section of `krb5.conf`.

**Return Values**

- None.
krb5_auth_con_free — Free auth_context

C Prototype

```c
krb5_error_code krb5_auth_con_free(
    krb5_context         context,
    krb5_auth_context    auth_context);
```

Arguments

- `context (input/output)`: The context structure.
- `auth_context (output)`: A per connection context.

Description

This routine frees the auth_context returned by `krb5_auth_con_init`.

Return Values

This routine returns the following KRB5 status code:

- `0`: Successful completion.
**C Prototype**

```c
krb5_error_code krb5_auth_con_genaddrs(
    krb5_context context,
    krb5_auth_context auth_context,
    int infd,
    int flags);
```

**Arguments**

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **infd (input)**: Input socket file descriptor.
- **flags (input)**: Input flags. These symbols are defined in `KRB$ROOT:[INCLUDE]KRB5.H`. The values can be OR'd together. Possible values for `flags` are:
  - `KRB5_AUTH_CONTEXT_GENERATE_LOCAL_ADDR`
  - `KRB5_AUTH_CONTEXT_GENERATE_REMOTE_ADDR`
  - `KRB5_AUTH_CONTEXT_GENERATE_LOCAL_FULL_ADDR`
  - `KRB5_AUTH_CONTEXT_GENERATE_REMOTE_FULL_ADDR`

**Description**

This routine takes an IP address and port, and generates a full IP address.

**Return Values**

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
**krb5_auth_con_get_checksum_func — Get the checksum function and data structure**

**C Prototype**

```c
krb5_error_code krb5_auth_con_get_checksum_func ( 
    krb5_context                    context, 
    krb5_auth_context               auth_context, 
    krb5_mk_req_checksum_func       *func, 
    void                            **data );
```

**Arguments**

- **context (input/output)** The context structure.
- **auth_context (input/output)** A per-connection context.
- **func (output)** A pointer to a function that performs the checksum.
- **data (output)** A pointer to the data structure that holds the checksum.

**Description**

This routine returns the checksum function and the data structure used to hold the checksum data.

**Return Values**

This routine returns the following KRB5 status code:

- **0** Successful completion.
krb5_auth_con_getrcache — Get the rcache element from the auth_context

C Prototype

```
krb5_error_code krb5_auth_con_getrcache (  
    krb5_context            context,  
    krb5_auth_context      auth_context,  
    krb5_rcache             rcache );
```

Arguments

- **context (input/output)**: The context structure.
- **auth_context (input)**: The authorization context.
- **rcache (output)**: The `rcache` element from the `auth_context`.

Description

This routine takes an IP address and port, and generates a full IP address.

Return Values

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
</tbody>
</table>
krb5_auth_con_getaddrs — Retrieve address fields from the auth_context

C Prototype

```c
krb5_error_code krb5_auth_con_getaddrs(
    krb5_context           context,
    krb5_auth_context      auth_context,
    krb5_address          **local_addr,
    krb5_address          **remote_addr );
```

Arguments

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **local_addr (output)**: Local address.
- **remote_addr (output)**: Remote address.

Description

This routine retrieves `local_addr` and `remote_addr` from `auth_context`. If `local_addr` or `remote_addr` is not NULL, the memory is first freed with `krb5_free_address` and then newly allocated. It is the caller's responsibility to free the returned addresses in this way.

Return Values

This routine returns the following KRB5 status code:

- **0**: Successful completion.
krb5_auth_con_getauthenticator — Retrieve authenticator used during mutual authentication

C Prototype

```c
krb5_error_code krb5_auth_con_getauthenticator(
    krb5_context context,
    krb5_auth_context auth_context,
    krb5_authenticator **authenticator );
```

Arguments

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **authenticator (output)**: The authenticator used during mutual authentication.

Description

This routine retrieves the authenticator that was used during mutual authentication. It is the caller's responsibility to free the memory allocated to authenticator by calling `krb5_free_authenticator`.

Return Values

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
</tbody>
</table>
**krb5_auth_con_getflags — Retrieve the flags in auth_context**

**C Prototype**

```c
krb5_error_code krb5_auth_con_getflags(
    krb5_context           context,
    krb5_auth_context      auth_context,
    krb5_int32             *flags );
```

**Arguments**

- **context (input/output)**: The context structure.
- **auth_context (input)**: A per connection context.
- **flags (input)**: A bit mask representing the flags to set in the `auth_context`. Valid flags are:
  - `KRB5_AUTH_CONTEXT_DO_TIME` — Use timestamps.
  - `KRB5_AUTH_CONTEXT_RET_TIME` — Save timestamps to output structure.
  - `KRB5_AUTH_CONTEXT_DO_SEQUENCE` — Use sequence numbers.
  - `KRB5_AUTH_RET_SEQUENCE` — Copy sequence numbers to output structure.

**Description**

This routine retrieves the flags from `auth_context`.

**Return Values**

This routine returns the following KRB5 status code:

- `0`: Successful completion.
**C Prototype**

```c
krb5_error_code krb5_auth_con_getkey(
    krb5_context context,
    krb5_auth_context auth_context,
    krb5_keyblock **keyblock );
```

**Arguments**

- **context** (input/output) The context structure.
- **auth_context** (input/output) A per-connection context.
- **keyblock** (output) Key stored in `auth_context`.

**Description**

This routine retrieves the keyblock stored in `auth_context`. The memory allocated in this function should be freed with a call to `krb5_free_keyblock`.

**Return Values**

This routine returns the following KRB5 status code:

- **0** Successful completion.
krb5_auth_con_getlocalseqnumber — Retrieve and store the local sequence number

C Prototype

```c
krb5_error_code krb5_auth_con_getlocalseqnumber(
    krb5_context           context,
    krb5_auth_context      auth_context,
    krb5_int32             *seqnumber );
```

Arguments

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **seqnumber (input)**: The address of the location to store the local sequence number.

Description

This routine retrieves the local sequence number that was used during authentication and stores it in `seqnumber`.

Return Values

This routine returns the following KRB5 status code:

- **0**: Successful completion.
krb5_auth_con_getrecvsubkey — Retrieve the recv_subkey keyblock from auth_context

C Prototype

```c
krb5_error_code krb5_auth_con_getrecvsubkey(
    krb5_context              context,
    krb5_auth_context         auth_context,
    krb5_keyblock             **keyblock );
```

Arguments

- **context** (input/output) The context structure.
- **auth_context** (input/output) A per-connection context.
- **keyblock** (output) recv_subkey keyblock stored in auth_context.

Description

This routine retrieves the recv_subkey keyblock stored in auth_context. The memory allocated in this function should be freed with a call to krb5_free_keyblock.

Return Values

This routine returns the following KRB5 status code:

```
0 Successful completion.
```
krb5_auth_con_getremoteseqnumber — Retrieve and store the remote sequence number

C Prototype

```c
krb5_error_code krb5_auth_con_getremoteseqnumber(
    krb5_context context,
    krb5_auth_context auth_context,
    krb5_int32 *seqnumber);
```

Arguments

- **context** (input/output) The context structure.
- **auth_context** (input/output) A per-connection context.
- **seqnumber** (input) The address of the location to store the remote sequence number.

Description

This routine retrieves the remote sequence number that was used during authentication and stores it in `seqnumber`.

Return Values

This routine returns the following KRB5 status code:

- 0 Successful completion.
**krb5_auth_con_getsendsubkey** — Retrieve the send_subkey keyblock from auth_context

**C Prototype**

```c
krb5_error_code krb5_auth_con_getsendsubkey(
    krb5_context context,
    krb5_auth_context auth_context,
    krb5_keyblock **keyblock);
```

**Arguments**

- **context** (input/output): The context structure.
- **auth_context** (input/output): A per-connection context.
- **keyblock** (output): send_subkey keyblock stored in auth_context.

**Description**

This routine retrieves the send_subkey keyblock stored in auth_context. The memory allocated in this function should be freed with a call to krb5_free_keyblock.

**Return Values**

This routine returns the following KRB5 status code:

- 0 Successful completion.
**krb5_auth_con_init — Initialize the auth_context**

**C Prototype**

```c
krb5_error_code krb5_auth_con_init(
    krb5_context        context,
    krb5_auth_context   *auth_context);
```

**Arguments**

- `context` (input/output) The context structure.
- `auth_context` (output) A per connection context.

**Description**

This routine initializes the `auth_context`. The `auth_context` contains all data pertinent to the various authentication routines.

The default flags for the context are set to enable the use of the replay cache (`krb5_auth_context_do_time`) but no sequence numbers. The function `krb5_auth_con_setflags` allows the flags to be changed.

The default checksum type is set to `CKSUMTYPE_RSA_MD4_DES`. This may be changed with `krb5_auth_con_setcksumtype`.

The `auth_context` structure should be freed with `krb5_auth_con_free`.

**Return Values**

This routine returns the following KRB5 status code:

```
0  Successful completion.
```
**krb5_auth_con_set_checksum_func — Set the checksum function and data structure**

**C Prototype**

```c
krb5_error_code krb5_auth_con_set_checksum_func(
    krb5_context context,
    krb5_auth_context auth_context,
    krb5_mk_req_checksum_func func,
    void *data);
```

**Arguments**

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **func (input)**: A function that performs the checksum.
- **data (input)**: A pointer to the data structure that holds the checksum.

**Description**

This routine sets the checksum function and the sets up the data structure used to hold the checksum data.

**Return Values**

This routine returns the following KRB5 status code:

- **0**: Successful completion.
**krb5_auth_con_setaddrs — Set address fields in auth_context**

**C Prototype**

```c
krb5_error_code krb5_auth_con_setaddrs(
    krb5_context           context,
    krb5_auth_context      auth_context,
    krb5_address           *local_addr,
    krb5_address           *remote_addr);
```

**Arguments**

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **local_addr (input)**: Local address.
- **remote_addr (input)**: Remote address.

**Description**

This routine copies the `local_addr` and `remote_addr` into `auth_context`. If either address is NULL, the previous address remains in place.

**Return Values**

This routine returns the following KRB5 status code:

- **0**: Successful completion.
krb5_auth_con_setflags — Set the flags in auth_context

C Prototype

```c
krb5_error_code krb5_auth_con_setflags(
    krb5_context       context,
    krb5_auth_context  auth_context,
    krb5_int32         flags );
```

Arguments

context (input/output)    The context structure.
auth_context (output)     A per-connection context.
flags (input)             A bit mask representing the flags to set in auth_context. Valid values are:
                          KRB5_AUTH_CONTEXT_DO_TIME — Use timestamps.
                          KRB5_AUTH_CONTEXT_RET_TIME — Save timestamps to output structure.
                          KRB5_AUTH_CONTEXT_DO_SEQUENCE — Use sequence numbers.
                          KRB5_AUTH_RET_SEQUENCE — Copy sequence numbers to output structure.

Description

This routine sets the flags of auth_context to the flags argument.

Return Values

This routine returns the following KRB5 status code:

```
0           Successful completion.
```
krb5_auth_con_setports — Set port fields in the auth_context

C Prototype

```c
krb5_error_code krb5_auth_con_setports(
    krb5_context context,
    krb5_auth_context auth_context,
    krb5_address *local_port,
    krb5_address *remote_port );
```

Arguments

- `context` (input/output): The context structure.
- `auth_context` (input/output): A per-connection context.
- `local_addr` (input): Local address.
- `remote_addr` (input): Remote address.

Description

This routine copies the `local_port` and `remote_port` addresses into `auth_context`. If either address is NULL, the previous address remains in place. These addresses are set by `krb5_auth_con_genaddrs`.

Return Values

This routine returns the following KRB5 status code:

- 0: Successful completion.
- ENOMEM: Insufficient memory.
krb5_auth_con_setrcache — Set the replay cache

C Prototype

```
krb5_error_code krb5_auth_con_setrcache(
    krb5_context context,
    krb5_auth_context auth_context,
    krb5_rcache rcache );
```

Arguments

context (input/output) The context structure.
auth_context (input/output) A per-connection context.
rcache (input) The replay cache to be set.

Description

This routine sets the replay cache that is used by the authentication routines to \texttt{rcache}.

Return Values

This routine returns the following KRB5 status code:

\begin{itemize}
  \item [0] Successful completion.
\end{itemize}
**krb5_auth_con_setrecvsubkey — Set the recv_subkey keyblock in auth_context**

**C Prototype**

```c
krb5_error_code krb5_auth_con_setrecvsubkey (
    krb5_context            context,
    krb5_auth_context       auth_context,
    krb5_keyblock           *keyblock );
```

**Arguments**

- `context (input/output)`: The context structure.
- `auth_context (input/output)`: A per-connection context.
- `keyblock (output)`: The `recv_subkey` keyblock stored in `auth_context`.

**Description**

This routine sets the `recv_subkey` keyblock that is stored in `auth_context`.

**Return Values**

This routine returns the following KRB5 status code:

- `0` Successful completion.
**krb5_auth_con_setsendsubkey — Set the send_subkey keyblock in auth_context**

**C Prototype**

```c
krb5_error_code krb5_auth_con_setsendsubkey (
    krb5_context            context,
    krb5_auth_context       auth_context,
    krb5_keyblock           *keyblock );
```

**Arguments**

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **keyblock (output)**: The `send_subkey` keyblock stored in `auth_context`.

**Description**

This routine sets the `send_subkey` keyblock that is stored in `auth_context`.

**Return Values**

This routine returns the following KRB5 status code:

- **0**: Successful completion.
krb5_auth_con_setuseruserkey — Set keyblock field in auth_context to temporary key

C Prototype

```c
krb5_error_code krb5_auth_con_setuseruserkey(
    krb5_context           context,
    krb5_auth_context      auth_context,
    krb5_keyblock          *keyblock);
```

Arguments

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **keyblock (input)**: Server key for incoming request.

Description

This routine overloads the keyblock field. It is only useful prior to a `krb5_rd_req_decode` call for user-to-user authentication where the server has the key and needs to use it to decrypt the incoming request. Once decrypted, this key is no longer necessary. It is then overwritten with the session key sent by the client.

Return Values

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Insufficient memory.</td>
</tr>
</tbody>
</table>
krb5_build_principal — Build a principal name

C Prototype

```
krb5_error_code krb5_build_principal(
    krb5_context     context,
    krb5_principal   *principal,
    int               rlen,
    const char        *realm,
    char              *s1, ...)
```

Arguments

- context (input/output) The context structure.
- principal (output) Principal name.
- rlen (input) Realm name length.
- realm (input) Realm name.
- ... (input) A variable-length argument list. These arguments are added to the principal data.

Description

This routine and krb5_build_principal_va perform the same function. krb5_build_principal takes a variable-length argument list, which is added to the principal data being built.

Both functions take a realm name realm, realm name length rlen, and a list of null-terminated strings, and fill in a pointer to a principal structure principal, making it point to a structure representing the named principal. The last string must be followed in the argument list by a NULL pointer.

Return Values

This routine returns the following KRB5 status code:

- 0 Successful completion.
- ENOMEM Insufficient memory.
krb5_build_principal_va — Fill in pointer to principal structure

C Prototype

```c
krb5_error_code krb5_build_principal_va(
    krb5_context    context,
    krb5_principal  princ,
    int             rlen,
    const char      *realm,
    va_list         ap );
```

Arguments

- **context (input/output)**: The context structure.
- **princ (output)**: The principal structure.
- **rlen (input)**: Realm name length.
- **realm (input)**: Realm name.
- **ap (input)**: A list of null-terminated strings.

Description

`krb5_build_principal` and `krb5_build_principal_va` perform the same function; the former takes variadic arguments, while the latter takes a pre-computed varargs pointer.

Both functions take a realm name region, realm name length `rlen`, and a list of null-terminated strings, and fill in a pointer to a principal structure `princ`, making it point to a structure representing the named principal. The last string must be followed in the argument list by a null pointer.

Return Values

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
**C Prototype**

```c
krb5_error_code krb5_c_block_size (  
    krb5_context            context,  
    krb5_enctype            enctype,  
    size_t                  *blocksize );
```

**Arguments**

- **context** (input/output): The context structure.
- **enctype** (input): The encryption type.
- **blocksize** (output): A pointer to the block size.

**Description**

This routine returns the block size for the encryption type `enctype` in the `blocksize` argument.

**Return Values**

This routine returns the following KRB5 status codes:

- 0: Successful completion.
- `KRB5_BAD_ENCTYPE`: Bad encryption type.
- `ENOMEM`: Insufficient memory.
**KRB5 (Kerberos V5) Application Programming Interface**

**krb5_c_checksum_length — Get the checksum length for a checksum type**

**C Prototype**

```c
krb5_error_code krb5_c_checksum_length (
    krb5_context context,
    krb5_cksumtype cksumtype,
    size_t *length );
```

**Arguments**

- **context (input/output)** The context structure.
- **cksumtype (input)** The checksum type.
- **length (output)** A pointer to the checksum length.

**Description**

This routine returns the checksum length for the checksum in `cksumtype` in the `length` argument.

**Return Values**

This routine returns the following KRB5 status codes:

- **0** Successful completion.
- **KRB5_BAD_ENCTYPE** Bad encryption type.
**krb5_c_decrypt — Decrypt encrypted data**

### C Prototype

```c
krb5_error_code krb5_c_decrypt (
    krb5_context context,
    const krb5_keyblock *key,
    krb5_keyusage usage,
    const krb5_data *ivec,
    const krb5_enc_data *input,
    krb5_data *output );
```

### Arguments

- **context** (input/output) The context structure.
- **key** (input) The key value from a keytab, ticket, etc.
- **usage** (input) A salt value.
- **ivec** (input) Input vector.
- **input** (input) The encrypted data.
- **output** (output) The decrypted data.

### Description

This routine decrypts encrypted data, given the proper key.

### Return Values

This routine returns the following KRB5 status code:

- KRB5_BAD_ENCTYPE Bad encryption type.
krb5_c_encrypt — Encrypt data

C Prototype

```c
krb5_error_code krb5_c_encrypt (  
    krb5_context            context,  
    const krb5_keyblock     *key,  
    krb5_keyusage           usage,  
    const krb5_data         *ivec,  
    const krb5_data         *input,  
    krb5_enc_data           *output );
```

Arguments

context (input/output)  The context structure.
key (input)  The key value from a keytab, ticket, etc.
usage (input)  A salt value.
ivec (input)  Input vector.
input (input)  The data to be encrypted.
output (output)  The encrypted data.

Description

This routine encrypts data with the given key.

Return Values

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRB5_BAD_ENCTYPE</td>
<td>Bad encryption type.</td>
</tr>
</tbody>
</table>

**krb5_c_encrypt_length** — Get the length of encrypted data

### C Prototype

```c
krb5_error_code krb5_c_encrypt_length(
    krb5_context context,
    krb5_enctype enctype,
    size_t inputlen,
    size_t *length);
```

### Arguments

- **context** (input/output) The context structure.
- **enctype** (input) The encryption type.
- **inputlen** (input) The length of the encrypted data to check.
- **length** (output) The length of the unencrypted data.

### Description

This routine finds the actual (unencrypted) length of data that has been encrypted. Encryption can potentially change the size of the data, so unencrypted and encrypted lengths may be different.

### Return Values

This routine returns the following KRB5 status code:

- **KRB5_BAD_ENCTYPE** Bad encryption type.
krb5_c_enctype_compare — Compare two encryption types

C Prototype

```c
krb5_error_code krb5_c_enctype_compare (
    krb5_context context,
    krb5_enctype e1,
    krb5_enctype e2,
    krb5_Boolean *similar );
```

Arguments

- **context** (input/output): The context structure.
- **e1** (input): First encryption type.
- **e2** (input): Second encryption type.
- **similar** (output): TRUE if types are similar; FALSE if types are different.

Description

This routine compares two encryption types.

Return Values

This routine returns the following KRB5 status code:

```
KRB5_BAD_ENCTYPE  Bad encryption type.
```
**krb5_c_is_coll_proof_cksum** — Test to see if a checksum is collision proof

**C Prototype**

```c
krb5_boolean krb5_c_is_coll_proof_cksum (
    const krb5_cksumtype ctype );
```

**Arguments**

cype (input) The checksum type to test.

**Description**

This routine tests the collision proof flag on the checksum given.

**Return Values**

This routine returns one of the following KRB5 status codes:

- 0 Checksum is not collision proof, or checksum type is not in the list.
- 1 Checksum is collision proof.
krb5_c_is_keyed_cksum — Test to see if a checksum uses derived keys

C Prototype

```c
krb5_boolean krb5_c_is_keyed_cksum (
    const krb5_cksumtype  ctype );
```

Arguments

- **ctype** (input) - The checksum type to test.

Description

This routine tests the derived flag for the checksum given.

Return Values

This routine returns one of the following KRB5 status codes:

- **0** - Checksum does not use derived keys, or checksum type is not in the list.
- **1** - Checksum uses derived keys.
krb5_c_keyed_checksum_types — Get a list of derived key checksums

C Prototype

```c
krb5_error_code krb5_c_keyed_checksum_types (  
    krb5_context context,  
    krb5_enctype enctype,  
    unsigned int *count,  
    krb5_cksumtype **cksumtypes );
```

Arguments

- `context` (input/output): The context structure.
- `enctype` (input): The encryption type.
- `count` (output): Pointer to a count of checksums matching the encryption type.
- `cksumtypes` (output): A pointer to the list of matching checksums.

Description

This routine searches the list of derived checksum types supported by Kerberos, and returns the list of checksum types matching the encryption type passed in `enctype` in the output parameter `cksumtypes`. The number of checksum types in `cksumtypes` is returned in `count`.

Return Values

This routine returns the following KRB5 status codes:

- `0`: Successful completion.
- `ENOMEM`: Insufficient memory.
**krb5_c_make_checksum — Compute a checksum**

**C Prototype**

```c
krb5_error_code krb5_c_make_checksum (  
    krb5_context context,  
    krb5_cksumtype cksumtype,  
    const krb5_keyblock *key,  
    krb5_keyusage usage,  
    const krb5_data *input,  
    krb5_checksum *cksum );
```

**Arguments**

- **context** (input/output) The context structure.
- **cksumtype** (input) The checksum type.
- **key** (input) A pointer to the encryption key.
- **usage** (input) A salt value.
- **input** (input) The data for which a checksum is to be produced.
- **cksum** (output) The checksum.

**Description**

This routine computes a checksum, which is returned in `cksum`. Input parameters include the checksum type `cksumtype`, the encryption key `key`, a salt value `usage`, and the data for which a checksum is to be produced in `input`.

**Return Values**

This routine returns the following KRB5 status codes:

- **0** Successful completion.
- **KRB5_BAD_ENCTYPE** Bad encryption type.
- **ENOMEM** Insufficient memory.
krb5_c_make_random_key — Generate a random key

C Prototype

```
krb5_error_code krb5_c_make_random_key (  
    krb5_context            context,  
    krb5_enctype            enctype,  
    krb5_keyblock           *random_key );
```

Arguments

- context (input/output) The context structure.
- enctype (input) The encryption type to use in generating the key.
- random_key (output) The random key.

Description

This routine generates a random key for a given encryption type.

Return Values

This routine returns the following KRB5 status codes:

- KRB5_BAD_ENCTYPE Bad encryption type.
- ENOMEM Insufficient memory.
**KRB5 (Kerberos V5) Application Programming Interface**

**krb5_c_random_make_octets — Create random data**

**C Prototype**

```c
krb5_error_code krb5_c_random_make_octets (
    krb5_context    context,
    krb5_data       *data );
```

**Arguments**

- **context (input/output)**: The context structure.
- **data (output)**: The random data.

**Description**

This routine creates random data using entropy from the Operating System.

**Return Values**

This routine returns the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>KRB5_CRYPTO_INTERNAL</td>
<td>Cryptosystem internal error.</td>
</tr>
</tbody>
</table>
krb5_c_random_seed — Get a random seed

C Prototype

```c
krb5_error_code Krb5_c_random_seed (  
    krb5_context    context,  
    krb5_data       *data );
```

Arguments

- **context** (input/output) The context structure.
- **data** (output) The random seed.

Description

This routine creates a random seed from sources of entropy available to the crypto routines.

Return Values

This routine returns the following KRB5 status codes:

- 0 Successful completion.
- `KRB5_CRYPTO_INTERNAL` Cryptosystem internal error.
krb5_c_string_to_key — Convert a string to a key

C Prototype

```c
krb5_error_code krb5_c_string_to_key (
    krb5_context            context,
    krb5_enctype            enctype,
    const krb5_data         *string,
    const krb5_data         *salt,
    krb5_keyblock           *key
);```

Arguments

- **context** (input/output) The context structure.
- **enctype** (input) The encryption type.
- **string** (input) The string to be converted.
- **salt** (input) The salt value.
- **key** (output) The generated key.

Description

This routine converts a string into a key, using the supplied encryption type and salt values.

Return Values

This routine returns the following KRB5 status codes:

- **KRB5_BAD_ENCTYPE** Bad encryption type.
- **KRB5_CRYPTO_INTERNAL** Cryptosystem internal error.
- **ENOMEM** Insufficient memory.
KRB5 (Kerberos V5) Application Programming Interface

**krb5_c_string_to_key_with_params** — Convert string key to keyblock

---

**C Prototype**

```c
krb5_error_code krb5_c_string_to_key_with_params (  
    krb5_context            context,  
    krb5_enctype            enctype,  
    const krb5_data         *string,  
    const krb5_data         *salt,  
    const krb5_data         *parsms,  
    krb5_keyblock           *key );
```

**Arguments**

- **context (input/output)**: The context structure.
- **enctype (input)**: The encryption type.
- **string (input)**: The string form of the key.
- **salt (input)**: The salt used in the encryption.
- **params (input)**: Special parameters used on the conversion.
- **key (output)**: The keyblock information.

**Description**

This routine converts a string key into a keyblock.

**Return Values**

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **KRBF5_BAD_ENCTYPE**: Bad encryption type.
- **KRBF5_CRYPTO_INTERNAL**: Cryptosystem internal error.
- **ENOMEM**: Insufficient memory.
krb5_c_valid_cksumtype — Validate a checksum type

C Prototype

krb5_boolean krb5_c_valid_cksumtype (  
    const krb5_cksumtype  ctype );

Arguments

ctype (input) The checksum type to validate.

Description

This routine tests to see whether a checksum type, passed in ctype, is a valid Kerberos checksum type.

Return Values

This routine returns one of the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Checksum type is invalid.</td>
</tr>
<tr>
<td>1</td>
<td>Checksum type is valid.</td>
</tr>
</tbody>
</table>
krb5_c_valid_enctype — Validate an encryption type

C Prototype

```
krb5_boolean krb5_c_valid_enctype (
    const krb5_enctype etype);
```

Arguments

- **etype (input)**: The encryption type to validate.

Description

This routine tests to see whether an encryption type, passed in `etype`, is a valid Kerberos encryption type.

Return Values

This routine returns one of the following KRB5 status codes:

- **0**: Encryption type is invalid.
- **1**: Encryption type is valid.
**krb5_c_verify_checksum — Verify a checksum**

**C Prototype**

```c
krb5_error_code krb5_c_verify_checksum (
    krb5_context context,
    const krb5_keyblock *key,
    krb5_keyusage usage,
    const krb5_data *data,
    const krb5_checksum *cksum,
    krb5_Boolean *valid );
```

**Arguments**

- `context` (input/output) — The context structure.
- `key` (input) — The key used to create the data in `cksum`.
- `usage` (input) — The key usage.
- `data` (input) — Data.
- `cksum` (input) — The checksum to verify.
- `valid` (output) — Non-zero if the checksum verified correctly; zero if it did not.

**Description**

This routine verifies the checksum of data in `cksum` that was created with a key using the key usage `usage`.

**Return Values**

This routine returns the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>KRB5_BAD_ENCTYPE</td>
<td>Bad encryption type.</td>
</tr>
<tr>
<td>KRB5_BAD_MSIZE</td>
<td>Message size is incompatible with encryption type.</td>
</tr>
</tbody>
</table>
krb5_cc_close — Close the credentials cache

C Prototype

```c
krb5_error_code krb5_cc_close(
    krb5_context      context,
    krb5_ccache       id );
```

Arguments

- `context` (input/output): The context structure.
- `id` (input/output): A credentials cache identifier.

Description

This routine closes the credentials cache `id`, invalidates `id`, and releases `id` and any other resources acquired during use of the credentials cache. It requires that `id` identifies a valid credentials cache. After return, `id` must not be used unless it is first reinitialized using `krb5_cc_resolve` or `krb5_cc_gen_new`.

Return Values

This routine returns the following KRB5 status code:

- Successful completion.
**C Prototype**

```c
krb5_error_code krb5_cc_copy_creds (  
    krb5_context            context,  
    krb5_ccache             incc,  
    krb5_ccache             outcc );
```

**Arguments**

- **context** (input/output): The context structure.
- **incc** (input): The credentials to be copied.
- **outcc** (output): The copy of the credentials.

**Description**

This routine creates a copy of the set of credentials found in `incc`. The copy is returned in `outcc`.

**Return Values**

This routine returns the following KRB5 status code:

- Kerberos errors.
**krb5_cc_default — Resolve the default credentials cache name**

**C Prototype**

```c
krb5_error_code krb5_cc_default(
    krb5_context context,
    krb5_ccache *ccache);
```

**Arguments**

- `context` (input/output): The context structure.
- `ccache` (output): A pointer to the credentials cache name.

**Description**

This routine is equivalent to `krb5_cc_resolve( context, krb5_cc_default_name(), ccache );`.

**Return Values**

This routine returns the following KRB5 status codes:

- `KV5M_CONTEXT` Bad magic number for `krb5_context` structure.

Results of `krb5_cc_resolve`
krb5_cc_default_name — Return the name of the default credentials cache

C Prototype

const char *krb5_cc_default_name(
    krb5_context context);

Arguments

custom (input/output) The context structure.

Description

This routine returns the name of the default credentials cache; this may be equivalent to
getenv(KRB5CCNAME) with an appropriate fallback.

Return Values

This routine returns the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>The name of the default credentials cache.</td>
</tr>
<tr>
<td>Failure</td>
<td>NULL</td>
</tr>
</tbody>
</table>
**krb5_cc_destroy — Destroy a credentials cache**

**C Prototype**

```c
krb5_error_code krb5_cc_destroy(
    krb5_context   context,
    krb5_ccache    id );
```

**Arguments**

- **context (input/output)**: The context structure.
- **id (input/output)**: A credentials cache identifier.

**Description**

This routine destroys the credentials cache identified by `id`, invalidates `id`, and releases any other resources acquired during use of the credentials cache. This routine requires that `id` identifies a valid credentials cache. After return, `id` must not be used unless it is first reinitialized using `krb5_cc_resolve` or `krb5_cc_gen_new`.

**Return Values**

This routine returns the following KRB5 status code:

- Permission errors.
**C Prototype**

```c
krb5_error_code krb5_cc_end_seq_get(
    krb5_context context,
    krb5_ccache id,
    krb5_cc_cursor *cursor );
```

**Arguments**

- **context** (input/output): The context structure.
- **id** (input/output): A credentials cache identifier.
- **cursor** (input/output): The cursor created by `krb5_cc_start_seq_get`.

**Description**

This routine finishes sequential processing mode and invalidates `*cursor`. `*cursor` must never be reused after this call.

It requires that `id` identifies a valid credentials cache and `*cursor` be a cursor returned by `krb5_cc_start_seq_get` or a subsequent call to `krb5_cc_next_cred`.

**Return Values**

This routine returns the following KRB5 status code:

```plaintext
    Error code if `*cursor` is invalid.
```
krb5_cc_gen_new — Generate a new credentials cache identifier

C Prototype

```c
krb5_error_code krb5_cc_gen_new(
    krb5_context          context,
    krb5_ccache           *id );
```

Arguments

context (input/output)    The context structure.
id (output)                A new, unique credentials cache identifier.

Description

This routine fills in `id` with a unique `ccache` identifier. The cache is left unopened.

Return Values

This routine returns the following KRB5 status code:

```
0                                Successful completion.
```
**krb5_cc_get_name — Return the name of the credentials cache**

**C Prototype**

```c
const char * krb5_cc_get_name(
    krb5_context context,
    krb5_ccache id);
```

**Arguments**

- **context (input/output)**: The context structure.
- **id (output)**: A credentials cache identifier.

**Description**

This routine returns the name of the credentials cache denoted by `id`.

**Return Values**

This routine returns the following KRB5 status code:

- **Success**: The name of the credentials cache specified as a second argument.
- **Failure**: NULL.
krb5_cc_get_principal — Retrieve the primary principal of the credentials cache

C Prototype

```c
krb5_error_code krb5_cc_get_principal(
    krb5_context     context,
    krb5_ccache      id,
    krb5_principal   *principal );
```

Arguments

- `context` (input/output): The context structure.
- `id` (input): A credentials cache identifier.
- `principal` (output): The returned primary principal.

Description

This routine retrieves the primary principal of the credentials cache (as set by `krb5_cc_initialize` request). The primary principal is set to `*principal`; the caller should release this memory by calling `krb5_free_principal` on `*principal` when finished.

It requires that `id` identifies a valid credentials cache.

Return Values

This routine returns the following KRB5 status code:

- Successful completion.
krb5_cc_get_type — Return the CC prefix

C Prototype

```c
const char *krb5_cc_get_type (  
    krb5_context    context,  
    krb5_ccache     cache );
```

Arguments

- `context` (input/output): The context structure.
- `cache` (input): The credentials cache.

Description

This routine returns the credentials cache prefix string as its return value.

Return Values

This routine returns the following:

A string representing the credentials cache prefix.
krb5_cc_initialize — Create/refresh a credentials cache

C Prototype

```c
krb5_error_code krb5_cc_initialize(
    krb5_context     context,
    krb5_ccache      id,
    krb5_principal   primary_principal);
```

Arguments

- **context (input/output)**: The context structure.
- **id (input/output)**: A credentials cache identifier.
- **primary_principal (input)**: The primary principal for the credentials cache.

Description

This routine creates or refreshes a credentials cache identified by `id` with the primary principal set to `primary_principal`. If the credentials cache already exists, its contents are destroyed. This routine also modifies cache identified by `id`.

Return Values

This routine returns one of the following KRB5 status codes:

- System errors.
- Permission errors.
krb5_cc_next_cred — Fetch the next credentials entry

C Prototype

```c
krb5_error_code krb5_cc_next_cred(
    krb5_context     context,
    krb5_ccache      id,
    krb5_cc_cursor   *cursor,
    krb5_creds       *creds );
```

Arguments

- **context (input/output)**: The context structure.
- **id (input/output)**: A credentials cache identifier.
- **cursor (input/output)**: The cursor created by `krb5_cc_start_seq_get`. This value is updated upon return to be used in subsequent calls to `krb5_cc_next_cred`. The returned credentials cache entry.
- **creds (output)**: The returned credentials cache entry.

Description

This routine fetches the next entry from `id`, returning its values in `*creds`, and updates `*cursor` for the next request. It requires that `id` identifies a valid credentials cache and `*cursor` is a cursor returned by `krb5_cc_start_seq_get` or a subsequent call to `krb5_cc_next_cred`. The `krb5_end_seq_get` routine is called when no more entries are to be read.

Return Values

This routine returns the following KRB5 status code:

- **Error code if there are no more cache entries.**
krb5_cc_remove_cred — Remove credentials from the credentials cache

C Prototype

```c
krb5_error_code krb5_cc_remove_cred(
    krb5_context     context,
    krb5_ccache      id,
    krb5_flags       which,
    krb5_creds       *cred );
```

Arguments

- context (input/output) The context structure.
- id (input) A credentials cache identifier.
- which (input) A bit mask representing the search flags to use. The values should be logically ORed together. Valid values are:
  - `KRB5_TC_MATCH_TIMES` – The requested lifetime is required to be at least as great as that specified.
  - `KRB5_TC_MATCH_IS_SKEY` – The `is_skey` field much match exactly.
  - `KRB5_TC_MATCH_FLAGS` – The set bits in `mcreds` must match in `creds`.
  - `KRB5_TC_MATCH_TIMES_EXACT` – The requested lifetime must match exactly.
  - `KRB5_TC_MATCH_FLAGS_EXACT` – All bits in `mcreds` must match exactly.
  - `KRB5_TC_MATCH_AUTHDATA` – The authorization data must match.
  - `KRB5_TC_MATCH_SRV_NAMEONLY` – Only the name portion of the principal name must match. The realm portion may be different. If this flag is not set, the entire principal name must match.
  - `KRB5_TC_MATCH_2ND_TKT` – The second tickets must match.
  - `KRB5_TC_MATCH_KTYPE` – The encryption key types must match.
  - `KRB5_TC_MATCH_SUPPORTED_KTYPES` – Check all matching entries that have any supported encryption type and return the one with the encryption type listed earliest. Return CC_NOT_KTYPE if a match is found except for having a supported encryption type.
- cred (input) The credentials to match.

Description

This routine removes any credentials from `id` which match the principal name (`cred->server`) and the fields in `cred` masked by `which`. It requires that `id` identifies a valid credentials cache.
Return Values

This routine returns one of the following KRB5 status codes:

- Error code if nothing matches.
- Error code if could not delete.
**krb5_cc_resolve — Resolve a credentials cache name**

**C Prototype**

```c
krb5_error_code krb5_cc_resolve(
    krb5_context   context,
    char           *string_name,
    krb5_ccache    *id );
```

**Arguments**

- **context (input/output)**
  The context structure.

- **string_name (input)**
  The credentials cache name to resolve.

- **id (output)**
  The credentials cache identifier that corresponds to the name in `string_name`.

**Description**

This routine fills in `id` with a ccache identifier that corresponds to the name in `string_name`. It requires that `string_name` be of the form `type=residual` and `type` is a type known to the library. Because of OpenVMS file naming differences, the `string_name` argument is formed in a slightly different way than on other platforms. The equal sign (=) is substituted for the colon (:) to separate the type from the residual.

**Return Values**

This routine returns the following KRB5 status code:

- **0**  Successful completion.
- **ENOMEM**  Insufficient memory.
**krb5_cc_retrieve_cred** — Search the cache for a credential and return it if found

### C Prototype

```c
krb5_error_code krb5_cc_retrieve_cred(
    krb5_context       context,
    krb5_ccache        id,
    krb5_flags         whichfields,
    krb5_creds         *mcreds,
    krb5_creds         *creds );
```

### Arguments

- **context** (input/output)  
The context structure.
- **id** (input)  
A credentials cache identifier.
- **whichfields** (input)  
A bit mask representing the search flags to use. The values should be logically ORed together. Valid values are:
  - `KRB5_TC_MATCH_TIMES` – The requested lifetime is required to be at least as great as that specified.
  - `KRB5_TC_MATCH_IS_SKEY` – The `is_skey` field must match exactly.
  - `KRB5_TC_MATCH_FLAGS` – The set bits in `mcreds` must match in `creds`.
  - `KRB5_TC_MATCH_TIMES_EXACT` – The requested lifetime must match exactly.
  - `KRB5_TC_MATCH_FLAGS_EXACT` – All bits in `mcreds` must match exactly.
  - `KRB5_TC_MATCH_AUTHDATA` – The authorization data must match.
  - `KRB5_TC_MATCH_SRV_NAMEONLY` – Only the name portion of the principal name must match. The realm portion may be different. If this flag is not set, the entire principal name must match.
  - `KRB5_TC_MATCH_2ND_TKT` – The second tickets must match.
  - `KRB5_TC_MATCH_KTYPE` – The encryption key types must match.
  - `KRB5_TC_MATCH_SUPPORTED_KTYPES` – Check all matching entries that have any supported encryption type and return the one with the encryption type listed earliest. Return CC_NOT_KTYPE if a match is found except for having a supported encryption type.
- **mcreds** (input)  
The credentials to match.
- **creds** (output)  
The credentials found in the cache that match the requested value.
Description

This routine searches the cache for credentials matching mcreds. The fields which are to be matched are specified by set bits in whichfields, and always include the principal name mcreds->server. This routine requires that id identifies a valid credentials cache.

If at least one match is found, one of the matching credentials is returned in *creds. The credentials should be freed using krb5_free_credentials.

Return Values

This routine returns the following KRB5 status code:

Error code if no matches found.
krb5_cc_set_default_name — Set default CC name

C Prototype

```c
krb5_error_code krb5_cc_set_default_name (
    krb5_context    context,
    const char      *name );
```

Arguments

cancel (input/output) The context structure.

name (input/output) The default credentials cache name.

Description

This routine sets the default credentials cache name. If the default name is not passed in the argument name, it defaults to the first valid entry of the following values: the KRB5CCNAME logical name, the file krb5cc_<PID> in a [.TMP] directory in the user’s login directory (where <PID> is the user’s process ID). If the KRB5CCNAME logical name is defined, it must not be a system-wide logical name.

Return Values

This routine returns the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>KV5M_CONTEXT</td>
<td>Bad magic number for krb5_context structure.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Insufficient memory.</td>
</tr>
</tbody>
</table>
**krb5_cc_set_flags — Set the flags on the credentials cache**

**C Prototype**

```c
krb5_error_code krb5_cc_set_flags(
    krb5_context context,
    krb5_ccache id,
    krb5_flags flags);
```

**Arguments**

- **context (input/output)**: The context structure.
- **id (input/output)**: A credentials cache identifier.
- **flags (input)**: A bit mask representing the flags to set. The values should be logically ORed together. Valid values are:
  - **KRB5_TC_OPENCLOSE**: Turn on OPENCLOSE mode (open and close the cache each time a credentials cache routine is called). The default, if this flag is not set, is to have the cache stay open until `krb5_cc_close` is called.

**Description**

This routine sets the flags on the credentials cache `id` to `flags`.

**Return Values**

This routine returns the following KRB5 status code:

- **Successful completion.**
krb5_cc_start_seq_get — Start sequential read of cached credentials

C Prototype

```c
krb5_error_code krb5_cc_start_seq_get(
    krb5_context    context,
    krb5_ccache     id,
    krb5_cc_curso   *cursor );
```

Arguments

- **context (input/output)** The context structure.
- **id (input)** A credentials cache identifier.
- **cursor (output)** A cursor to be used in calls to krb5_cc_next_cred.

Description

This routine prepares to sequentially read every set of cached credentials.

Return Values

This routine returns the following KRB5 status code:

- **Successful completion.**
**krb5_cc_store_cred — Store a credential in the credentials cache**

**C Prototype**

```c
krb5_error_code krb5_cc_store_cred(
    krb5_context     context,
    krb5_ccache      id,
    krb5_creds       *creds );
```

**Arguments**

- **context (input/output)** The context structure.
- **id (input)** A credentials cache identifier.
- **creds (input)** The credentials to store in the cache.

**Description**

This routine stores creds in the cache `id`, tagged with `creds->client`. It requires that `id` identifies a valid credentials cache.

**Return Values**

This routine returns one of the following KRB5 status codes:

- Permission error.
- Storage failure error.
krb5_change_password — Change an existing password

C Prototype

```c
krb5_error_code krb5_change_password (  
    krb5_context context,  
    krb5_creds *creds,  
    char *newpw,  
    int *result_code,  
    krb5_data *result_code_string,  
    krb5_data *result_string );
```

Arguments

- `context` (input/output): The context structure.
- `creds` (input): The Kerberos credentials.
- `newpw` (input): The new password.
- `result_code` (output): A numeric error code.
- `result_code_string` (output): The string equivalent of the result_code.
- `result_string` (output): The change password response from the KDC.

Description

This routine changes the password for an existing Kerberos account.

Return Values

This routine returns one of the following KRB5 status codes:

- 0: Successful completion.
- KRB5KRB_AP_ERR_MODIFIED: Message stream modified.
- KRB5KDC_ERR_BAD_PVNO: Requested protocol version not supported.
- ENOMEM: Insufficient memory.
- SOCKET_ERRNO: Error on socket.
- ETIMEDOUT: Connection timed out.
- EHOSTUNREACH: No route to host.
krb5_cksumtype_to_string — Convert checksum type to string representation

C Prototype

krb5_error_code krb5_cksumtype_to_string ( krb5_cksumtype cksumtype, char *buffer, size_t buflen );

Arguments

cksumtype (input)   The checksum type to convert.
buffer (output)     A pointer to a buffer to hold the string value of the checksum type.
buflen (input)      The maximum string length that can fit in buffer.

Description

This routine changes the password for an existing Kerberos account.

Return Values

This routine returns one of the following KRB5 status codes:

- 0  Successful completion.
- ENOMEM  Insufficient memory.
- EINVAL  Invalid argument.
krb5_copy_addresses — Copy Kerberos addresses

C Prototype

krb5_error_code krb5_copy_addresses(
    krb5_context           context,
    krb5_address * const   *inaddr,
    krb5_address           ***outaddr );

Arguments

custom (input/output) The context structure.
inaddr (input) An array of addresses.
outaddr (output) A pointer to a copy of the array of addresses.

Description

This routine copies addresses in inaddr to *outaddr, which is allocated memory and should be freed with krb5_free_addresses.

Return Values

This routine returns the following KRB5 status code:

    0                Successful completion.
    ENOMEM          Insufficient memory.
KRB5 (Kerberos V5) Application Programming Interface

**krb5_copy_authdata — Copy a Kerberos authdata structure**

**C Prototype**

```c
krb5_error_code krb5_copy_authdata(
    krb5_context           context,
    krb5_authdata * const  *inauthdat,
    krb5_authdata          ***outauthdat );
```

**Arguments**

- **context (input/output)** The context structure.
- **inauthdat (input)** An array of krb5_authdata structures. The last element must be NULL.
- **outauthdat (output)** A copy of the array of krb5_authdata structures.

**Description**

This routine copies an authdata structure, filling in *outauthdat to point to the newly allocated copy, which should be freed with krb5_free_authdata.

**Return Values**

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Insufficient memory.</td>
</tr>
</tbody>
</table>
 krb5_copy_authenticator — Copy an authenticator structure

C Prototype

```
krb5_error_code krb5_copy_authenticator(
    krb5_context                   context,
    const krb5_authenticator       *authfrom,
    krb5_authenticator             **authto );
```

Arguments

- **context (input/output)**: The context structure.
- **authfrom (input)**: The authenticator to be copied.
- **authto (output)**: A copy of the authenticator.

Description

This routine copies an authenticator structure, filling in *outauthdat to point to the newly allocated copy, which should be freed with krb5_free_authenticator.

Return Values

This routine returns the following KRB5 status code:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
krb5_copy_checksum — Copy a checksum structure

C Prototype

```
krb5_error_code krb5_copy_checksum(
    krb5_context           context,
    const krb5_checksum    *ckfrom,
    krb5_checksum          **ckto );
```

Arguments

- **context (input/output)**: The context structure.
- **ckfrom (input)**: The checksum to be copied.
- **ckto (output)**: A pointer to a copy of the checksum.

Description

This routine copies a checksum structure, filling in *ckto to point to the newly allocated copy, which should be freed with krb5_free_checksum.

Return Values

This routine returns the following KRB5 status code:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
**krb5_copy_creds — Copy a credentials structure**

**C Prototype**

```c
krb5_error_code krb5_copy_creds(
    krb5_context context,
    const krb5_creds *incred,
    krb5_creds **outcred );
```

**Arguments**

- **context** (input/output) The context structure.
- **incred** (input) The credentials structure to be copied.
- **outcred** (output) A pointer to a copy of the credentials structure.

**Description**

This routine copies a credentials structure, filling in `*outcred` to point to the newly allocated copy, which should be freed with `krb5_free_creds`.

**Return Values**

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Insufficient memory.</td>
</tr>
</tbody>
</table>
krb5_copy_data — Copy a Kerberos data structure

C Prototype

```c
krb5_error_code krb5_copy_data(
    krb5_context context,
    const krb5_data *indata,
    krb5_data **outdata);
```

Arguments

- **context** (input/output): The context structure.
- **indata** (input): The data structure to be copied.
- **outdata** (output): A pointer to a copy of the data structure.

Description

This routine copies a data structure, filling in `*outdata` to point to the newly allocated copy, which should be freed with `krb5_free_data`.

Return Values

This routine returns the following KRB5 status code:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
krb5_copy_keyblock — Copy a keyblock

C Prototype

#include <krb5.h>

krb5_error_code krb5_copy_keyblock(
    krb5_context            context,
    const krb5_key lock    *from,
    krb5_keyblock          **to );

Arguments

context (input/output) The context structure.
from (input)            The keyblock to copy.
to (output)             A pointer to a copy of the keyblock.

Description

This routine copies a keyblock, and sets the *to argument to point to the newly allocated copy, which should be freed with krb5_free_keyblock.

Return Values

This routine returns the following KRB5 status code:

0                      Successful completion.
ENOMEM                  Insufficient memory.
krb5_copy_keyblock_contents — Copy a keyblock’s contents

C Prototype

```c
krb5_error_code krb5_copy_keyblock_contents(
    krb5_context context,
    const krb5_keyblock *from,
    krb5_keyblock *to );
```

Arguments

- **context (input/output)**: The context structure.
- **from (input)**: The keyblock to copy the contents of.
- **to (output)**: A pointer to a copy of the keyblock contents.

Description

This routine copies keyblock contents from `from` to `to`, including allocated storage. The allocated storage should be freed by using `free(to->contents)`.

Return Values

This routine returns the following KRB5 status code:

- 0: Successful completion.
- ENOMEM: Insufficient memory.
**krb5_copy_principal — Copy a principal structure**

**C Prototype**

```c
krb5_error_code krb5_copy_principal(
    krb5_context             context,
    krb5_const_principal     inprinc,
    krb5_principal           *outprinc );
```

**Arguments**

- **context (input/output)** The context structure.
- **inprinc (input)** Principal name to be copied.
- **outprinc (output)** Copy of input principal name.

**Description**

This routine copies a principal structure, setting *outprinc to point to the newly allocated copy, which should be freed with krb5_free_principal.

**Return Values**

This routine returns the following KRB5 status code:

- **0** Successful completion.
- **ENOMEM** Insufficient memory.
krb5_copy_ticket — Copy a Kerberos ticket structure

C Prototype

```c
krb5_error_code krb5_copy_ticket(
    krb5_context context,
    const krb5_ticket *from,
    krb5_ticket **pto);
```

Arguments

- **context** (input/output): The context structure.
- **from** (input): The ticket structure to be copied.
- **pto** (output): A pointer to a copy of the ticket structure.

Description

This routine copies a ticket structure, setting *pto to point to the newly allocated copy, which should be freed with `krb5_free_ticket`.

Return Values

This routine returns the following KRB5 status code:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
KRB5 (Kerberos V5) Application Programming Interface

**krb5_decode_ticket — Decode a formatted ticket**

**C Prototype**

```c
krb5_error_code krb5_decode_ticket (
    const krb5_data *code,
    krb5_ticket **rep );
```

**Arguments**

- **code (input)**: The formatted ticket.
- **rep (output)**: The decoded ticket information.

**Description**

This routine takes a formatted ticket code and decodes it, filling in `rep` with the results.

The contents of `rep` are set to allocated storage that should be freed by the caller (using `krb5_free_ticket`) when finished with the ticket.

**Return Values**

This routine returns the following KRB5 status codes:

- 0: Successful completion.
- KRB5KDC_ERR_BAD_PVNO: Bad key version number.
krb5_deltat_to_string — Convert a Kerberos relative time value to a string

C Prototype

```c
krb5_error_code krb5_deltat_to_string (
    krb5_deltat deltat,
    char *buffer,
    size_t buflen);
```

Arguments

deltat (input)  The relative time value to convert.
buffer (output) A pointer to a buffer to hold the time string.
buflen (input)  The maximum length of the string that will fit in buffer.

Description

This routine converts a Kerberos relative time value into the corresponding string.

Return Values

This routine returns the following KRB5 status codes:

- 0  Successful completion.
- ENOMEM  Insufficient memory.
**krb5_enctype_to_string — Convert a Kerberos encryption type value to a string**

**C Prototype**

```c
krb5_error_code krb5_enctype_to_string (
    krb5_enctype enctype,
    char *buffer,
    size_t buflen );
```

**Arguments**

- **enctype (input)**: The encryption type value to convert.
- **buffer (output)**: A pointer to a buffer to hold the encryption type string.
- **buflen (input)**: The maximum length of the string that will fit in buffer.

**Description**

This routine converts a Kerberos encryption type value into the corresponding string.

**Return Values**

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
krb5_free_addresses — Free a group of addresses

C Prototype

```c
void krb5_free_addresses(
    krb5_context context,
    krb5_address **val );
```

Arguments

- **context** (input/output) The context structure.
- **val** (input/output) A pointer to the data structure to be freed.

Description

This routine frees a series of addresses `val` that have been previously allocated via Kerberos APIs such as `krb5_copy_addresses`.

Return Values

This routine returns the following KRB5 status code:

None.
**krb5_free_ap_rep_enc_part — Free subkey and other data allocated by krb5_rd_rep or krb5_send_auth**

**C Prototype**

```c
void krb5_free_ap_rep_enc_part(
    krb5_context           context,
    krb5_ap_rep_enc_part   *val );
```

**Arguments**

- **context (input/output)** The context structure.
- **val (input/output)** A pointer to the data structure to be freed.

**Description**

This routine frees the subkey keyblock (if set) as well as `val` that has been allocated from `krb5_rd_rep` or `krb5_send_auth`.

**Return Values**

This routine returns the following KRB5 status code:

None.
krb5_free_authdata — Free an authdata structure

C Prototype

```c
void krb5_free_authdata(
    krb5_context context,
    krb5_authdata **val );
```

Arguments

- **context** (input/output): The context structure.
- **val** (input/output): A pointer to the data structure to be freed.

Description

This routine frees the authdata structure pointed to by **val** that has been allocated from `krb5_copy_authdata`.

Return Values

This routine returns the following KRB5 status code:

- **None**.
**krb5_free_authenticator — Free authenticator storage**

### C Prototype

```c
void krb5_free_authenticator(
    krb5_context context,
    krb5_authenticator *val );
```

### Arguments

- **context (input/output)** The context structure.
- **val (input/output)** A pointer to the data structure to be freed.

### Description

This routine frees the authenticator `val`, including the pointer `val`.

### Return Values

This routine returns the following KRB5 status code:

- None.
**krb5_free_checksum — Free a checksum**

**C Prototype**

```c
void krb5_free_checksum(
    krb5_context     context,
    krb5_checksum    *val );
```

**Arguments**

- `context` (input/output): The context structure.
- `val` (input/output): A pointer to the data structure to be freed.

**Description**

This routine frees checksum and the pointer `val`.

**Return Values**

This routine returns the following KRB5 status code:

None.
krb5_free_checksum_contents — Free the contents of a checksum structure

C Prototype

```c
void krb5_free_checksum_contents (
    krb5_context            context,
    register krb5_checksum  *val );
```

Arguments

- `context` (input/output) The context structure.
- `val` (input/output) The checksum value to free.

Description

This routine frees the contents of a Kerberos checksum structure.

Return Values

None.
**krb5_free_cksumtypes — Free a checksum structure**

**C Prototype**

```c
void krb5_free_cksumtypes ( 
    krb5_context    context, 
    krb5_cksumtype  *val );
```

**Arguments**

- **context (input/output)**: The context structure.
- **val (input/output)**: The checksum type to free.

**Description**

This routine frees a Kerberos checksum structure `val`. The contents of the structure should have already been freed.

**Return Values**

None.
krb5_free_context — Free a context structure

C Prototype

Void krb5_free_context(
    krb5_context  context );

Arguments

context (input)           Context structure to be freed.

Description

This routine frees the context returned by krb5_init_context. Internally calls krb5_os_free_context.

Return Values

None.
krb5_free_creds — Free credentials

C Prototype

```c
void krb5_free_creds(
    krb5_context context,
    krb5_creds *val);
```

Arguments

- `context (input/output)` The context structure.
- `val (input/output)` A pointer to the data structure to be freed.

Description

This routine calls `krb5_free_cred_contents` with `val` as the argument. `val` is freed as well.

Return Values

This routine returns the following KRB5 status code:

```
None.
```
krb5_free_cred_contents — Free credential structures

C Prototype

```c
void krb5_free_cred_contents(
    krb5_context context,
    krb5_creds *val);
```

Arguments

- **context (input/output)**: The context structure.
- **val (input/output)**: A pointer to the data structure to be freed.

Description

This routine zeros out the session key stored in the credential and then frees the credentials structures. The argument **val** is not freed.

Return Values

This routine returns the following KRB5 status code:

- None.
**krb5_free_data — Free storage associated with a krb5_data object**

**C Prototype**

```c
void krb5_free_data(
    krb5_context    context,
    krb5_data       *val );
```

**Arguments**

- `context` (input/output) The context structure.
- `val` (input/output) A pointer to the data structure to be freed.

**Description**

This routine frees the data structure `val`, including the pointer `val`, which has been allocated by any of numerous routines.

**Return Values**

This routine returns the following KRB5 status code:

None.
**krb5_free_data_contents — Frees contents of a krb5_data structure**

**C Prototype**

```c
void krb5_free_data_contents (            
    krb5_context            context,    
    krb5_data               *val );
```

**Arguments**

- `context` (input/output) — The context structure.
- `val` (input/output) — The krb5_data structure to be freed.

**Description**

This routine frees the contents of a krb5_data structure, and sets the data field in the structure to zero.

**Return Values**

This routine returns the following KRB5 status code:

- `None`. 
ктb5_free_default_realm — Free the Kerberos default realm structure

C Prototype

```c
void krb5_free_default_realm (
    krb5_context    context,
    char            *lrealm );
```

Arguments

- `context` (input/output) The context structure.
- `lrealm` (input/output) The realm structure to be freed.

Description

This routine frees the Kerberos default realm structure.

Return Values

None.
krb5_free_error — Free error information

C Prototype

```c
void krb5_free_error(
    krb5_context   context,
    krb5_error     *val);
```

Arguments

- **context (input/output)**: The context structure.
- **val (input/output)**: A pointer to the data structure to be freed.

Description

This routine frees the error `val` that has been allocated from `krb5_read_error` or `krb5_sendauth`.

Return Values

This routine returns the following KRB5 status code:

None.
**C Prototype**

```c
krb5_error_code krb5_free_host_realm(
    krb5_context     context,
    char * const     *realmlist);
```

**Arguments**

- **context** (input) The context structure.
- **realmlist** (output) A pointer to a list of realm names.

**Description**

This routine frees the storage taken by a `realmlist` returned by `krb5_get_host_realm`.

**Return Values**

This routine returns the following KRB5 status code:

```
None.
```
**krb5_free_keyblock — Free keyblock memory**

**C Prototype**

```c
void krb5_free_keyblock(
    krb5_context     context,
    krb5_keyblock    *val);
```

**Arguments**

- **context** (input/output) The context structure.
- **val** (input/output) A pointer to the data structure to be freed.

**Description**

This routine frees the pointer `val` and memory, and zeroes the keyblock contents of `val`.

**Return Values**

This routine returns the following KRB5 status code:

- None.
krb5_free_keyblock_contents — Free the contents of a key structure

C Prototype

void krb5_free_keyblock_contents (
    krb5_context context,
    register krb5_keyblock key );

Arguments

context (input/output)  The context structure.
key (input/output)     The key structure whose contents are to be freed.

Description

This routine frees the contents of a Kerberos keyblock structure.

Return Values

None.
**krb5_free_keytab_entry_contents — Free the contents of a keytab entry**

**C Prototype**

```c
krb5_error_code krb5_free_keytab_entry_contents (  
    krb5_context            context,  
    krb5_keytab_entry       *entry );
```

**Arguments**

- **context (input/output)** The context structure.
- **entry (input/output)** The keytab entry whose contents are to be freed.

**Description**

This routine frees the contents of a Kerberos keytab entry.

**Return Values**

This routine returns the following KRB5 status code:

0 Successful completion.
**krb5_free_principal — Free the pwd_data allocated by krb5_copy_principal**

### C Prototype

```c
void krb5_free_principal(
    krb5_context context,
    krb5_principal val);
```

### Arguments

- **context (input/output)** The context structure.
- **val (input/output)** A pointer to the data structure to be freed.

### Description

This routine frees the `pwd_data` `val` that has been allocated from `krb5_copy_principal`.

### Return Values

This routine returns the following KRB5 status code:

- **None.**
**krb5_free_tgt_creds — Free TGT credentials**

**C Prototype**

```c
void krb5_free_tgt_creds(
    krb5_context     context,
    krb5_creds       **tgts );
```

**Arguments**

- **context (input/output)**: The context structure.
- **tgts (input/output)**: A pointer to the credentials to be freed.

**Description**

This routine frees the TGT credentials `tgts` returned by `krb5_get_cred_from_kdc`.

**Return Values**

This routine returns the following KRB5 status code:

- **None**.
krb5_free_ticket — Free ticket allocated by krb5_copy_ticket

C Prototype

```c
void krb5_free_ticket(
    krb5_context     context,
    krb5_ticket      *val );
```

Arguments

countext (input/output) The context structure.
val (input/output) A pointer to the data structure to be freed.

Description

This routine frees the ticket val that has been allocated from krb5_copy_ticket and other routines.

Return Values

This routine returns the following KRB5 status code:

None.
**krb5_free_unparsed_name — Free a simple name**

**C Prototype**

```c
void krb5_free_unparsed_name ( 
    krb5_context context, 
    char *val );
```

**Arguments**

- **context** (input/output) The context structure.
- **val** (input/output) The name string to be freed.

**Description**

This routine frees the memory associated with a simple character string name.

**Return Values**

This routine returns the following KRB5 status code:

None.
**krb5_fwd_tgt_creds — Get a TGT for use at a remote host**

### C Prototype

```c
krb5_error_code krb5_fwd_tgt_creds (  
  krb5_context            context,  
  krb5_auth_context       auth_context,  
  char                    *rhost,  
  krb5_principal          client,  
  krb5_principal          server,  
  krb5_ccache             cc,  
  int                     forwardable,  
  krb5_data               *outbuf );
```

### Arguments

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: A per-connection context.
- **rhost (input/output)**: The remote host.
- **client (input)**: The client principal.
- **server (input)**: The server principal.
- **cc (input)**: The credentials cache name.
- **forwardable (input)**: A Boolean indicating whether the TGT should be forwardable.
- **outbuf (output)**: The output buffer containing the TGT.

### Description

This routine acquires a TGT for use at a remote host system.

### Return Values

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
- **KRB5_PRINC_NOMATCH**: Requested principal and ticket don’t match.
- **KRB5_NO_TKT_SUPPLIED**: Request did not supply a ticket.
- **KRB5_CC_BADNAME**: Credential cache name or principal name malformed.
krb5_get_credentials — Get an additional ticket for the client

C Prototype

```c
krb5_error_code krb5_get_credentials(
    krb5_context           context,
    const krb5_flags       options,
    krb5_ccache            ccache,
    krb5_creds             *in_creds,
    krb5_creds             *out_creds );
```

Arguments

- **context (input/output)** The context structure.
- **options (input)** Valid values are as follows:
  - `KRB5_GC_USER_USER` — Return a full user to user authentication ticket
  - `KRB5_GC_CACHED` — Only search credentials cache for the ticket.
- **ccache (input)** The credentials cache.
- **in_creds (input)** Input credentials.
- **out_creds (output)** Output credentials.

Description

This routine attempts to use the credentials cache `ccache` or a TGS exchange to get an additional ticket for the client identified by `in_creds->client`, with the following information:

- The server identified by `in_creds->server`.
- The options in `options`. Valid choices are `KRB5_GC_USER_USER` and `KRB5_GC_CACHED`.
- The expiration date specified in `in_creds->times.endtime`.
- The session key type specified in `in_creds->keyblock.keytype` if it is nonzero.

If `options` specifies `KRB5_GC_CACHED`, then `krb5_get_credentials` will only search the credentials cache for a ticket.

If `options` specifies `KRB5_GC_USER_USER`, then `krb5_get_credentials` will get credentials for a user-to-user authentication. In a user-to-user authentication, the secret key for the server is the session key from the server's ticket granting ticket (TGT). The TGT is passed from the server to the client over the network; this is safe since the TGT is encrypted in a key known only by the Kerberos server. The client must pass this TGT to `krb5_get_credentials` in `in_creds->second_ticket`. The Kerberos server will use this TGT to construct a user-to-user ticket that can be verified by the server, by using the session key from its TGT.

The effective expiration date is the minimum of the following:

- The expiration date as specified in `in_creds->times.endtime`.
- The requested start time plus the maximum lifetime of the server as specified by the server's entry in the Kerberos database.
The requested start time plus the maximum lifetime of tickets allowed in the local site, as specified by the KDC. This is a compile-time option, `KRB5_KDB_MAX_LIFE` in `config.h`, and is by default one day.

If any special authorization data needs to be included in the ticket for example, restrictions on how the ticket can be used, they should be specified in `in_creds->authdata`. If there is no special authorization data to be passed, `in_creds->authdata` should be NULL.

Any returned ticket and intermediate ticket-granting tickets are stored in `ccache`.

**Return Values**

This routine returns one of the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Insufficient memory.</td>
</tr>
</tbody>
</table>
**krb5_get_credentials_renew — Renew a set of existing credentials**

**C Prototype**

```c
krb5_error_code krb5_get_credentials_renew ( 
    krb5_context context, 
    krb5_flags options, 
    krb5_ccache ccache, 
    krb5_creds *in_creds, 
    krb5_creds **out_creds );
```

**Arguments**

- `context (input/output)` The context structure.
- `options (input)` Unused flag field.
- `ccache (input/output)` The credentials cache.
- `in_creds (input)` The credentials to be renewed.
- `out_creds (output)` The refreshed credentials.

**Description**

This routine attempts to contact a KDC to renew a set of existing Kerberos credentials.

**Return Values**

This routine returns one of the following KRB5 status codes:

- `0` Successful completion.
- `ENOMEM` Insufficient memory.
- `KRB5_PROG_ETYPE_NOSUPP` Program lacks support for encryption type.
- `KRB5_KDCREP_MODIFIED` KDC reply did not match expectations.
- Kerberos errors.
krb5_get_credentials_validate — Validate a set of existing credentials

C Prototype

```c
krb5_error_code krb5_get_credentials_validate (
    krb5_context            context,
    krb5_flags              options,
    krb5_ccache             ccache,
    krb5_creds              *in_creds,
    krb5_creds              **out_creds );
```

Arguments

- context (input/output): The context structure.
- options (input): Unused flag field.
- ccache (input/output): The credentials cache.
- in_creds (input): The credentials to be validated.
- out_creds (output): The validated credentials.

Description

This routine attempts to contact a KDC to validate a set of existing Kerberos credentials.

Return Values

This routine returns one of the following KRB5 status codes:

- 0: Successful completion.
- ENOMEM: Insufficient memory.
- KRB5_PROG_ETYPE_NOSUPP: Program lacks support for encryption type.
- KRB5_KDCREP_MODIFIED: KDC reply did not match expectations.
- Kerberos errors.
krb5_get_default_realm— Retrieve the default realm

C Prototype

```c
krb5_error_code krb5_get_default_realm(
    krb5_context context,
    char **lrealm);
```

Arguments

- context (input): The context structure.
- lrealm (output): A pointer to the default realm.

Description

This routine retrieves the default realm to be used if no user-specified realm is available (for example, to interpret a user-typed principal name with the realm omitted for convenience), setting `lrealm` with a pointer to the default realm in allocated storage.

It is the caller's responsibility for freeing the allocated storage pointed to be `lrealm` when it is finished with it.

Return Values

This routine returns the following KRB5 status code:

- System errors.
KRB5 (Kerberos V5) Application Programming Interface

krb5_get_init_creds_keytab — Get initial credentials’ keytab

C Prototype

```c
krb5_error_code krb5_get_init_creds_keytab (  
    krb5_context context,  
    krb5_creds *creds,  
    krb5_principal client,  
    krb5_keytab arg_keytab,  
    krb5_deltat start_time,  
    char *in_tkt_service,  
    krb5_get_init_creds_opt *options);
```

Arguments

- `context` (input/output): The context structure.
- `creds` (output): A pointer to a Kerberos credentials structure.
- `client` (input): The client principal.
- `arg_keytab` (input): A keytab handle.
- `start_time` (input): The time when the ticket becomes valid.
- `in_tkt_service` (input): The principal name of the requesting server.
- `options` (input): A pointer to a structure containing flags and options.

Description

This routine gets the keytab associated with the initial credentials. This may be either the default context’s keytab, or the keytab of the client credentials.

Return Values

This routine returns the following KRB5 status codes:

- `0`: Successful completion.
- `ENOMEM`: Insufficient memory.
krb5_get_init_creds_opt_init — Initialize options for krb5_get_init_creds* routines

C Prototype

void krb5_get_init_creds_opt_init (    
    krb5_get_init_creds_opt *opt );

Arguments

opt (input/output)  A pointer to a structure containing flags and options.

Description

This routine sets the flags field of the krb5_get_init_creds_opt structure to zero.

Return Values

None.
krb5_get_init_creds_opt_set_address_list — Set the address list in krb5_get_init_creds_opt

C Prototype

```c
void krb5_get_init_creds_opt_set_address_list (  
    krb5_get_init_creds_opt         *opt,  
    krb5_address                    **addresses );
```

Arguments

- **opt (output)**: A pointer to the options field.
- **addresses (input)**: A pointer to the address to set in the address list field in `opt`.

Description

This routine sets the address list in the `krb5_get_init_creds_opt` structure.

Return Values

- None.
**krb5_get_init_creds_opt_set_etype_list — Set the encryption list field in krb5_get_init_creds_opt**

**C Prototype**

```c
void krb5_get_init_creds_opt_set_etype_list ( 
    krb5_get_init_creds_opt         *opt, 
    krb5_enctype                    *etype_list, 
    int                             etype_list_length );
```

**Arguments**

- **opt (output)** A pointer to the options field.
- **etype_list (input)** A pointer to the encryption type to set.
- **etype_list_length (input)** The length of the etype_list field.

**Description**

This routine sets the encryption list field in the krb5_get_init_creds_opt structure.

**Return Values**

None.
krb5_get_init_creds_opt_set_forwardable — Set the forwardable field in krb5_get_init_creds_opt

C Prototype

```c
void krb5_get_init_creds_opt_set_forwardable (
    krb5_get_init_creds_opt     *opt,
    int                         forwardable );
```

Arguments

- `opt` (output) A pointer to the options field.
- `forwardable` (input) A flag indicating whether the credentials are forwardable.

Description

This routine sets the forwardable field in the `krb5_get_init_creds_opt` structure.

Return Values

None.
**krb5_get_init_creds_opt_set_preauth_list — Set the preauth_list field in krb5_get_init_creds_opt**

**C Prototype**

```c
void krb5_get_init_creds_opt_set_preauth_list (  
    krb5_get_init_creds_opt *opt,  
    krb5_preauthtype *preauth_list,  
    int preauth_list_length );
```

**Arguments**

- `opt` (output) A pointer to the options field.
- `preauth_list` (input) A pointer to the preauthentication type to set.
- `preauth_list_length` (input) The length of the `preauth_list` field.

**Description**

This routine sets the `preauth_list` field in the `krb5_get_init_creds_opt` structure.

**Return Values**

None.
krb5_get_init_creds_opt_set_proxiable — Set the proxiable field in krb5_get_init_creds_opt

C Prototype

```c
void krb5_get_init_creds_opt_set_proxiable (  
    krb5_get_init_creds_opt *opt,  
    int proxiable );
```

Arguments

- **opt (output)**: A pointer to the options field.
- **proxiable (input)**: A flag indicating whether the credentials are proxiable.

Description

This routine sets the proxiable field in the `krb5_get_init_creds_opt` structure.

Return Values

None.
**KRB5 (Kerberos V5) Application Programming Interface**

**krb5_get_init_creds_opt_set_renew_life — Set the renewal lifetime field in krb5_get_init_creds_opt**

**C Prototype**

```c
void krb5_get_init_creds_opt_set_renew_life (
    krb5_get_init_creds_opt         *opt,
    krb5_deltat                     renew_life );
```

**Arguments**

- `opt` (output) A pointer to the options field.
- `renew_life` (input) The Kerberos ticket renewal lifetime.

**Description**

This routine sets the Kerberos ticket renewal lifetime field in the `krb5_get_init_creds_opt` structure.

**Return Values**

- None.
krb5_get_init_creds_opt_set_salt — Set the salt field in krb5_get_init_creds_opt

C Prototype

```c
void krb5_get_init_creds_opt_set_salt (    krb5_get_init_creds_opt         *opt,
                                         krb5_data                       *salt );
```

Arguments

- **opt** (output) A pointer to the options field.
- **salt** (input) A pointer to the salt value.

Description

This routine sets the cryptographic salt field in the krb5_get_init_creds_opt structure.

Return Values

None.
krb5_get_init_creds_opt_set_tkt_life — Initialize the ticket lifetime for krb5_get_init_creds* routines

C Prototype

```c
void krb5_get_init_creds_opt_set_tkt_life (  
    krb5_get_init_creds_opt     *opt,  
    krb5_deltat                 tkt_life );
```

Arguments

- **opt (input/output)**: A pointer to a structure containing flags and options.
- **tkt_life (input)**: The ticket lifetime.

Description

This routine initializes the ticket lifetime information in preparation for calling krb5_get_init_creds* routines. It sets the ticket lifetime flag in the options flag field, and initializes the ticket lifetime in `opt` to `tkt_life`.

Return Values

None.
krb5_get_init_creds_password — Get the initial credentials password

C Prototype

krb5_error_code krb5_get_init_creds_password (  
    krb5_context context,  
    krb5_creds *creds,  
    krb5_principal client,  
    char *password,  
    krb5_prompter_fct prompter,  
    void *data,  
    krb5_deltat start_time,  
    char *in_tkt_service,  
    krb5_get_init_creds_opt *options );

Arguments

context (input/output) The context structure.
creds (output) A pointer to a Kerberos credentials structure.
client (input) The client principal.
password (input/output) The password associated with the initial credentials.
prompter (input) A pointer to a password prompt routine.
data (input) The data for the password prompt routine.
start_time (input) The time that the credentials first become valid.
in_tkt_service (input) A pointer to the output buffer containing the TGT.
options (input) A pointer to a structure containing flags and options.

Description

This routine acquires the password associated with the initial credentials.

Return Values

This routine returns the following KRB5 status codes:

None. Successful completion.
EINVAL Invalid argument.
KR5_KDC_UNREACH Cannot contact any KDC for requested realm.
KR5_PREAUTH_FAILED Generic preauthentication failure.
KR5_LIBOS_PWDINTR Password read interrupted.
KR5_REALM_CANT_RESOLVE Cannot resolve network address for KDC in requested realm.
KR5KDC_ERR_KEY_EXP Password has expired.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRB5_LIBOS_BADPWDMATCH</td>
<td>Password mismatch.</td>
</tr>
<tr>
<td>KRB5_CHPW_PWDNULL</td>
<td>New password cannot be zero length.</td>
</tr>
<tr>
<td>KRB5_CHPW_FAIL</td>
<td>Password change failed.</td>
</tr>
</tbody>
</table>
**krb5_get_host_realm — Get the Kerberos realm names for a host**

**C Prototype**

```c
krb5_error_code krb5_get_host_realm(
    krb5_context context,
    const char *host,
    char ***realmlist);
```

**Arguments**

- **context** (input): The context structure.
- **host** (input): The host name.
- **realmlist** (output): A pointer to a list of realm names.

**Description**

This routine determines the Kerberos realm names for `host`, filling in `realmlist` with a pointer to an `argv[]` style list of names, terminated with a NULL pointer.

If `host` is NULL, the local host's realms are determined.

If there are no known realms for the host, the filled-in pointer is set to NULL.

The pointer array and strings pointed to are all in allocated storage, and should be freed by the caller when finished.

**Return Values**

This routine returns the following KRB5 status code:

- 0: Successful completion.
- ENOMEM: Insufficient memory.
krb5_get_message — Convert an error code into the string representation

C Prototype

char * krb5_get_message(
    long code);

Arguments

code (input) The Kerberos numeric error code.

Description

This routine is supported on the OpenVMS platform only. It converts a Kerberos numeric error code into the string that describes the error.

Return Values

A pointer to an ASCII string describing the error indicated by code. The storage allocated at this pointer location should not be freed; it is part of an internal table of error messages.
**krb5_get_permitted_enctypes — Return a list of supported encryption types**

**C Prototype**

```c
krb5_error_code krb5_get_permitted_enctypes (  
    krb5_context            context,  
    krb5_enctype            **ktypes );
```

**Arguments**

- `context` (input/output) The context structure.
- `ktypes` (output) A pointer to the list of encryption types.

**Description**

This routine returns a list of the supported encryption types.

**Return Values**

This routine returns the following KRB5 status codes:

- **0** Successful completion.
- **ENOMEM** Insufficient memory.
- **KRB5_CONFIGETYPE_NOSUPP** No valid encryption types configured.
krb5_get_prompt_types — Get prompt_types from the Kerberos context

C Prototype

```c
krb5_prompt_type * krb5_get_prompt_types (  
    krb5_context context );
```

Arguments

context (input/output) The context structure.

Description

This routine returns the prompt_types field from the Kerberos context structure.

Return Values

A pointer to the krb5_prompt_type field, which contains one of the following values:

- KRB5_PROMPT_TYPE_PASSWORD
- KRB5_PROMPT_TYPE_NEW_PASSWORD
- KRB5_PROMPT_TYPE_NEW_PASSWORD_AGAIN
- KRB5_PROMPT_TYPE_PREAUTH
krb5_get_renewed_creds — Renew existing credentials

C Prototype

```c
krb5_error_code krb5_get_renewed_creds (
    krb5_context            context,
    krb5_creds              *creds,
    krb5_principal          client,
    krb5_ccache             ccache,
    char                    *in_tkt_service );
```

Arguments

context (input/output) The context structure.
creds (output) A pointer to a Kerberos credentials structure.
client (input) The client principal.
ccache (input) The credentials cache name.
in_tkt_service (input) A pointer to the principal name of the requesting server.

Description

This routine renews a set of existing Kerberos credentials.

Return Values

This routine returns the following KRB5 status codes:

- 0 Successful completion.
- ENOMEM Insufficient memory.
- KRB5_PROG_ETYPE_NOSUPP Program lacks support for encryption type.
C Prototype

```c
krb5_error_code krb5_get_server_rcache(
    krb5_context           context,
    const krb5_data        *piece,
    krb5_rcache            *ret_rcache );
```

Arguments

- **context** (input/output): The context structure.
- **piece** (input): Used to distinguish this replay cache from others in use on the system. Typically, `piece` is the first component of the principal name for the client or server that is calling `krb5_get_server_rcache`.
- **ret_recache** (output): A handle to an open `rcache`.

Description

This routine generates a replay cache name, allocates space for its handle, and opens it.

Upon successful return, `ret_recache` is filled in to contain a handle to an open `rcache`, which should be closed with `krb5_rc_close`.

Return Values

This routine returns the following KRB5 status code:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
krb5_get_time_offsets — Get the time offsets from the os context

C Prototype

```c
krb5_error_code krb5_get_time_offsets (  
    krb5_context            context,  
    krb5_int32              *seconds,  
    krb5_int32              *microseconds );
```

Arguments

- **context (input/output)**: The context structure.
- **seconds (output)**: A pointer to os context time_offset.
- **microseconds (output)**: A pointer to the os context usec_offset.

Description

This routine returns the second and microsecond time offsets from the os context.

Return Values

This routine returns the following KRB5 status code:

- `0` Successful completion.
KRB5 (Kerberos V5) Application Programming Interface

krb5_get_validated_creds — Get validated credentials

C Prototype

```c
krb5_error_code krb5_get_validated_creds (  
    krb5_context context,  
    krb5_creds *creds,  
    krb5_principal client,  
    krb5_ccache ccache,  
    char *in_tkt_service );
```

Arguments

- `context` (input/output) The context structure.
- `creds` (output) A pointer to a Kerberos credentials structure.
- `client` (input) The client principal.
- `ccache` (input) The credentials cache name.
- `in_tkt_service` (input) A pointer to the principal name of the requesting server.

Description

This routine acquires a set of validated credentials from the KDC.

Return Values

This routine returns the following KRB5 status codes:

- `0` Successful completion.
- `KRB5_NO_2ND_TKT` Request missing second ticket.
- `KRB5_NO_TKT_SUPPLIED` Request did not supply a ticket.
- `KRB5_PRINC_NOMATCH` Requested principal and ticket don’t match.
- `KRB5_KDCREP_MODIFIED` KDC reply did not match expectations.
- `KRB5_KDCREP_SKEW` Clock skew too great in KDC reply.
- `ENOMEM` Insufficient memory.
**krb5_init_context — Initialize a Kerberos context structure**

**C Prototype**

```c
krb5_error_code krb5_init_context(
    krb5_context *context);
```

**Arguments**

context (output) A pointer to the context structure that has been initialized.

**Description**

This routine initializes the context for the application. The context contains the encryption types, a pointer to operating specific data and the default realm. In the future, the context may also contain thread specific data. The data in the context should be freed with `krb5_free_context`.

**Return Values**

This routine returns the following KRB5 status code:

0 Successful completion.

ENOMEM Insufficient memory.
KRB5 (Kerberos V5) Application Programming Interface

krb5_init_keyblock — Set up an empty keyblock

C Prototype

```c
krb5_error_code krb5_init_keyblock (
    krb5_context            context,
    krb5_enctype            enctype,
    size_t                  length,
    krb5_keyblock           **out );
```

Arguments

- `context` (input/output): The context structure.
- `enctype` (input): The encryption type.
- `length` (input): The length of the keyblock.
- `out` (output): A pointer to the empty keyblock.

Description

This routine sets up an empty keyblock.

Return Values

This routine returns the following KRB5 status codes:

- `0`: Successful completion.
- `ENOMEM`: Insufficient memory.
**krb5_init_secure_context — Initialize a secure Kerberos context block**

**C Prototype**

```c
krb5_error_code krb5_init_secure_context (
    krb5_context *context);
```

**Arguments**
- `context` (output) A pointer to the context structure to be initialized.

**Description**
This routine initializes a secure Kerberos context block, preparing it for use by other Kerberos APIs.

**Return Values**
This routine returns the following KRB5 status codes:

- 0 Successful completion.
- ENOMEM Insufficient memory.
- Kerberos errors.
**krb5_is_thread_safe — Check whether the Kerberos client code supports multithreading**

**C Prototype**

```c
krb5_boolean krb5_is_thread_safe ( void );
```

**Description**

This routine returns a value indication whether the Kerberos client libraries (KRB$RTL.EXE, KRB$RTL32.EXE) are thread safe. On OpenVMS as of OpenVMS V8.3 (Kerberos V3.0), multithreading support is always enabled.

**Return Values**

This routine returns the following KRB5 status codes:

- `TRUE`: Client libraries are thread safe.
- `FALSE`: Client libraries are not thread safe.
**krb5_kt_add_entry — Add an entry to a key table**

**C Prototype**

```c
krb5_error_code krb5_kt_add_entry(
    krb5_context context,
    krb5_keytab id,
    krb5_keytab_entry *entry );
```

**Arguments**

- **context (input/output)**: The context structure.
- **id (input)**: A key table handle.
- **entry (input)**: The new entry to add to the key table.

**Description**

This routine adds a new entry to a key table. If the table is not writeable, then KRB5_KT_NOWRITE is returned.

**Return Values**

This routine returns the following KRB5 status code:

- 0: Successful completion.
- ENOMEM: Insufficient memory.
krb5_kt_close — Close a key table

C Prototype

```c
krb5_error_code krb5_kt_close(
    krb5_context context,
    krb5_keytab id );
```

Arguments

- context (input/output) The context structure.
- id (input/output) A key table handle.

Description

This routine closes the keytab identified by id and invalidates id, and releases any other resources acquired during use of the key table.

It requires that id identifies a keytab.

Return Values

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
</tbody>
</table>
krb5_kt_default — Return a handle to the default keytab

C Prototype

```c
krb5_error_code krb5_kt_default(
    krb5_context     context
    krb5_keytab      *id);
```

Arguments

custom (input/output) The context structure.
id (input/output) A key table handle.

Description

This routine fills `id` with a handle identifying the default keytab.

Return Values

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
</tbody>
</table>
krb5_kt_default_name — Get default key table name

C Prototype

```c
krb5_error_code krb5_kt_default_name(
    krb5_context context,
    char *name,
    int namesize);
```

Arguments

- **context** (input/output) The context structure.
- **name** (input/output) Key table name to resolve.
- **namesize** (input) The size of the name to return. Anything more than namesize will be zeroed in name upon completion.

Description

This routine fills `name` with the first `namesize` bytes of the name of the default keytab. If the name is shorter than `namesize`, then the remainder of `name` will be zeroed.

Return Values

This routine returns the following KRB5 status code:

- `0` Successful completion.
krb5_kt_end_seq_get — Complete a series of sequential key table entry retrievals

C Prototype

```c
krb5_error_code krb5_kt_end_seq_get(
    krb5_context context,
    krb5_keytab id,
    krb5_kt_cursor *cursor);
```

Arguments

- `context` (input/output): The context structure.
- `id` (input/output): A key table handle.
- `cursor` (input/output): The cursor to be invalidated.

Description

This routine finishes sequential processing mode and invalidates `cursor`, which must never be reused after this routine call.

This routine requires that `id` identifies a valid keytab and `*cursor` be a cursor returned by `krb5_kt_start_seq_get` or a subsequent call to `krb5_kt_next_entry`.

Return Values

This routine returns the following KRB5 status code:

- 0: Successful completion.
**krb5_kt_get_entry — Retrieve an entry from the key table**

**C Prototype**

```c
krb5_error_code krb5_kt_get_entry(
    krb5_context context,
    krb5_keytab id,
    krb5_principal principal,
    krb5_kvno vno,
    krb5_keytype keytype,
    krb5_keytab_entry *entry);
```

**Arguments**

- **context** (input/output) The context structure.
- **id** (input/output) A key table handle.
- **principal** (input) A principal name.
- **vno** (input) Key version number. If `vno` is zero, the first entry whose principal matches is returned.
- **keytype** (input) The key encryption type. Use a `keytype` of zero if an encryption type does not matter.
- **entry** (output) The returned key table entry.

**Description**

This routine searches the keytab identified by `id` for an entry whose principal matches `principal`, whose keytype matches `keytype`, and whose key version number matches `vno`. It returns an error code if no suitable entry is found. If an entry is found, the entry is returned in `*entry`; its contents should be deallocated by calling `krb5_kt_free_entry` when no longer needed.

**Return Values**

This routine returns the following KRB5 status code:

- 0 Successful completion.
**krb5_kt_get_name — Get key table name**

**C Prototype**

```c
krb5_error_code krb5_kt_get_name(
    krb5_context context,
    krb5_keytab id,
    char *name,
    unsigned int namesize);
```

**Arguments**

- **context (input/output)**: The context structure.
- **id (input/output)**: A key table handle.
- **name (output)**: The key table name.
- **namesize (input)**: The maximum length to fill in name.

**Description**

This routine fills `name` with the first `namesize` bytes of the name of the keytab identified by `id`. If the name is shorter than `namesize`, then `name` will be NULL terminated.

**Return Values**

This routine returns the following KRB5 status code:

- **0**: Successful completion.
**krb5_kt_get_type — Return the keytab prefix**

**C Prototype**

```c
char * krb5_kt_get_type ( 
    krb5_context    context, 
    krb5_keytab     keytab );
```

**Arguments**

- `context (input/output)` The context structure.
- `keytab (output)` The keytab structure whose type is returned.

**Description**

This routine returns the keytab prefix string as its return value.

**Return Values**

This routine returns the following:

- A string representing the prefix value of the keytab structure.
krb5 _kt_next_entry — Retrieve the next entry from the key table

C Prototype

krb5 _error _code krb5 _kt_next_entry (  
    krb5 _context context,  
    krb5 _keytab id,  
    krb5 _keytab _entry *entry,  
    krb5 _kt _cursor *cursor );

Arguments

dcontext (input/output) The context structure.

did (input/output) A key table handle.

dentry (output) The returned key table entry.

dcursor (input/output) A cursor to be used in subsequent calls to krb5 _kt_next_entry .

Description

This routine fetches the next entry in the keytab, returning it in *entry, and updates *cursor for the next request. If the keytab changes during the sequential get, an error is guaranteed. The argument *entry should be freed after use by calling krb5 _kt_free_entry .

This routine requires that id identifies a valid keytab, and *cursor be a cursor returned by krb5 _kt_start_seq_get or a subsequent call to krb5 _kt_next_entry .

Return Values

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
</tbody>
</table>
krb5_kt_read_service_key — Retrieve a service key from the key table

C Prototype

```c
krb5_error_code krb5_kt_read_service_key( 
    krb5_context     context,
    krb5_pointer     keyprocarg,
    krb5_principal   principal,
    krb5_kvno        vno,
    krb5_keytype     keytype,
    krb5_keyblock    **key );
```

Arguments

- **context** (input/output) The context structure.
- **keyprocarg** (input)  The name of a keytab, or NULL to use the default keytab.
- **principal** (input)  The service principal.
- **vno** (input)  Key version number. Use a vno of zero to specify the key with the highest version number.
- **keytype** (input) The key encryption type. Use a keytype of zero if an encryption type does not matter.
- **key** (output)  The returned service key.

Description

The routine opens and searches keytab for the entry identified by principal, keytype, and vno, returning the resulting key in *key or returning an error code if it is not found. If keyprocarg is not NULL, it is taken to be a char* denoting the name of a keytab. Otherwise, the default keytab will be used.

`krb5_free_keyblock` should be called on *key when the caller is finished with the key.

Return Values

This routine returns the following KRB5 status code:

- 0 Successful completion.
krb5_kt_remove_entry — Remove an entry from a key table

C Prototype

```
krb5_error_code krb5_kt_remove_entry(
    krb5_context         context,
    krb5_keytab          id,
    krb5_keytab_entry    *entry );
```

Arguments

- **context** (input/output) The context structure.
- **id** (input) A key table handle.
- **entry** (input) The entry to remove from the key table.

Description

This routine removes an entry from a key table. If this routine is not available, then KRB5_KT_NOWRITE is returned.

Return Values

This routine returns the following KRB5 status code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>KRB5_KT_NOWRITE</td>
<td>Keytab is not writable.</td>
</tr>
</tbody>
</table>
**C Prototype**

```c
krb5_error_code krb5_kt_resolve(
    krb5_context    context,
    const char      *string_name,
    krb5_keytab     *id );
```

**Arguments**

- `context` (input/output) The context structure.
- `string_name` (input) The name of the keytab.
- `id` (output) A keytab handle.

**Description**

This routine fills in `id` with a handle identifying the keytab with the name `string_name`. The keytab is not opened. The routine requires that `string_name` be of the form `type:residual` and `type` is a type known to the library.

**Return Values**

This routine returns the following KRB5 status code:

- `0` Successful completion.
- `ENOMEM` Insufficient memory.
- `KRB5_KT_UNKNOWN_TYPE` Unknown keytab type.
krb5_kt_start_seq_get — Start a sequential retrieve of key table entries

C Prototype

```c
krb5_error_code krb5_kt_start_seq_get(
    krb5_context       context,
    krb5_keytab        id,
    krb5_kt_cursor     *cursor );
```

Arguments

- **context** (input/output) The context structure.
- **id** (input/output) A key table handle.
- **cursor** (output) A cursor to be used in calls to krb5_kt_next_entry.

Description

This routine prepares to read sequentially every key in the keytab identified by `id`. The `cursor` argument is filled in with a cursor to be used in calls to `krb5_kt_next_entry`.

Return Values

This routine returns the following KRB5 status code:

- **0** Successful completion.
**krb5_kuserok — Determine whether the local user is authorized to log in**

**C Prototype**

```c
krb5_boolean krb5_kuserok(
    krb5_context context,
    krb5_principal principal,
    const char *luser );
```

**Arguments**

- **context (input)**: The context structure.
- **principal (input)**: A Kerberos principal name.
- **luser (input)**: A local username.

**Description**

This routine determines whether user is authorized to log in to the account `luser`, given a Kerberos principal `principal` and a local username `luser`.

**Return Values**

This routine returns one of the following KRB5 status codes:

- **TRUE**: User is authorized to log in.
- **FALSE**: User is not authorized to log in.
KRB5 (Kerberos V5) Application Programming Interface

krb5_mk_1cred — Encode a KRB_CRED message for krb5_rd_cred

C Prototype

```
krb5_error_code krb5_mk_1cred (  
    krb5_context            context,  
    krb5_auth_context       auth_context,  
    krb5_creds              *pcreds,  
    krb5_data               **ppdata,  
    krb5_replay_data        *outdata );
```

Arguments

- context (input/output): The context structure.
- auth_context (input): The Kerberos authentication context.
- pcreds (input): A pointer to Kerberos credentials.
- ppdata (input): A pointer to a `krb5_data` structure (not used).
- outdata (output): A pointer to the KRB_CRED message.

Description

This routine takes a Kerberos credential, and returns a KRB_CRED message in `outdata` that is suitable for `krb5_rd_cred`. This is a convenience function that calls `krb5_mk_ncred` with only a single set of credentials.

Return Values

This routine returns the following KRB5 status codes:

- 0: Successful completion.
- ENOMEM: Insufficient memory.
- KRB5_RC_REQUIRED: Message replay detection requires `rcache` parameter.
**C Prototype**

```c
krb5_error_code krb5_mk_error(
    krb5_context context,
    const krb5_error *dec_err,
    krb5_data *enc_err);
```

**Arguments**

- **context** (input/output): The context structure.
- **dec_err** (input): The error structure to format.
- **enc_err** (output): The formatted error buffer.

**Description**

This routine formats the error structure `*dec_err` into an error buffer `*enc_err`.

The error buffer storage (`enc_err->data`) is allocated, and should be freed by the caller when finished.

**Return Values**

This routine returns the following KRB5 status code:

- 0: Successful completion.
C Prototype

```c
krb5_error_code krb5_mk_ncred (  
    krb5_context            context,  
    krb5_auth_context       auth_context,  
    krb5_creds              **ppcreds,  
    krb5_data               **ppdata,  
    krb5_replay_data        *outdata );
```

Arguments

- **context (input/output)**: The context structure.
- **auth_context (input)**: The Kerberos authentication context.
- **ppcreds (input)**: A pointer to an array of Kerberos credentials.
- **ppdata (input)**: A pointer to a `krb5_data` structure (not used).
- **outdata (output)**: A pointer to the KRB_CRED message.

Description

This routine takes an array of Kerberos credentials, and returns a KRB_CRED message in `outdata` that is suitable for `krb5_rd_cred`.

Return Values

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
- **KRB5_RC_REQUIRED**: Message replay detection requires `rcache` parameter.
krb5_mk_priv — Format a KRB_PRIV message

C Prototype

```c
krb5_error_code krb5_mk_priv(
    krb5_context context,
    krb5_auth_context auth_context,
    const krb5_data *userdata,
    krb5_data *outbuf,
    krb5_replay_data *outdata );
```

Arguments

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: Authentication context. The flags from `auth_context` select whether sequence numbers or timestamps should be used to identify the message. Valid values are:
  - `KRB5_AUTH_CONTEXT_DO_TIME` — Use timestamps and replay cache.
  - `KRB5_AUTH_CONTEXT_RET_TIME` — Copy timestamp to `*outdata`.
  - `KRB5_AUTH_CONTEXT_DO_SEQUENCE` — Use sequence numbers in replay cache.
  - `KRB5_AUTH_CONTEXT_RET_SEQUENCE` — Use sequence numbers in replay cache and output data.
- **userdata (input)**: The user data in the message.
- **outbuf (output)**: The formatted KRB_PRIV buffer.
- **outdata (input/output)**: Contains the sequence numbers if `KRB5_AUTH_CONTEXT_RET_SEQUENCE` was specified in `auth_context`.

Description

This routine formats a KRB_PRIV message into `outbuf`. Behaves similarly to `krb5_mk_safe`, but the message is encrypted and integrity protected rather than just integrity-protected.

The `inbuf`, `auth_context`, `outdata` and `outbuf` arguments function as in `krb5_mk_safe`.

As in `krb5_mk_safe`, the `remote_addr` and `remote_port` part of the `auth_context` is optional; if the receiver's address is not known, it may be replaced by NULL. The `local_addr`, however, is mandatory.

The encryption type is taken from the `auth_context` keyblock portion. If the `i_vector` portion of the `auth_context` is nonNULL, it is used as an initialization vector for the encryption (if the chosen encryption type supports initialization vectors), and its contents are replaced with the last block of encrypted data upon return.
**Return Values**

This routine returns one of the following KRB5 status codes:

\[
\begin{array}{ll}
0 & \text{Successful completion.}
\end{array}
\]
krb5_mk_rep — Format and encrypt an AP_REP message

C Prototype

```c
krb5_error_code krb5_mk_rep(
    krb5_context          context,
    krb5_auth_context     auth_context,
    krb5_data             *outbuf );
```

Arguments

- context (input/output) The context structure.
- auth_context (input/output) Authentication context.
- outbuf (output) AP_REQ message information.

Description

This routine formats and encrypts an AP_REP message, including in it the data in the authentp portion of auth_context, encrypted using the keyblock portion of auth_context.

When successful, outbuf->length and outbuf->data are filled in with the length of the AP_REQ message and allocated data holding it. The outbuf->data argument should be freed by the caller when it is no longer needed.

If the flags in auth_context indicate that a sequence number should be used (either KRB5_AUTH_CONTEXT_DO_SEQUENCE or KRB5_AUTH_CONTEXT_RET_SEQUENCE) and the local sequence number in the auth_context is 0, a new number will be generated with krb5_generate_seq_number.

Return Values

This routine returns the following KRB5 status code:

- 0 Successful completion.
KRB5 (Kerberos V5) Application Programming Interface

**krb5 mk_req — Format a KRB_AP_REQ message**

**C Prototype**

```c
krb5_error_code krb5_mk_req(
    krb5_context context,
    krb5_auth_context *auth_context,
    const krb5_flags ap_req_options,
    char *service,
    char *hostname,
    krb5_data *in_data,
    krb5_ccache ccache,
    krb5_data *outbuf );
```

**Arguments**

- **context (input/output)** The context structure.
- **auth_context (input/output)** Authentication context. Contains the checksum method to be used. A new authentication context will be returned if NULL is specified.
- **ap_req_options (input)** Specifies the KRB_AP_REQ options desired. Valid options are:
  - **AP_OPTS_USE_SESSION_KEY**
  - **AP_OPTS_MUTUAL_REQUIRED**
  - **AP_OPTS_USE_SUBKEY**
- **service (input)** Used to specify the principal name, in conjunction with hostname.
- **hostname (input)** The server to receive the message.
- **in_data (input)** Application data whose checksum should be included in the authenticator. Specify NULL if no checksum is to be included.
- **ccache (input/output)** The credentials cache.
- **outbuf (output)** A pointer to an existing krb5_data structure to be filled. Returns the generated AP_REQ message.

**Description**

This routine formats a KRB_AP_REQ message into *outbuf*.

The principal of the server to receive the message is specified by *hostname* and *service*. If credentials are not present in the credentials cache *ccache* for this server, the TGS request with default arguments is used in an attempt to obtain such credentials, and they are stored in *ccache*.

The checksum method to be used is as specified in *auth_context*.

The *outbuf* argument should point to an existing krb5_data structure. *outbuf->length* and *outbuf->data* will be filled in on success, and the latter should be freed by the caller when it is no longer needed; if an error is returned, however, no storage is allocated and *outbuf->data* does not need to be freed.
Return Values

This routine returns one of the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
</tbody>
</table>
krb5_mk_req_extended — Format a KRB_AP_REQ message with additional options

C Prototype

```c
krb5_error_code krb5_mk_req_extended(
    krb5_context           context,
    krb5_auth_context      *auth_context,
    const krb5_flags       ap_req_options,
    krb5_data              *in_data,
    krb5_creds             *in_creds,
    krb5_data              *outbuf );
```

Arguments

context (input/output)  The context structure.
auth_context (input/output)  Authentication context. Contains the checksum method to be used. A new authentication context will be returned if NULL is specified.
ap_req_options (input)  Specifies the KRB_AP_REQ options desired. Valid options are:
  AP_OPTS_USE_SESSION_KEY
  AP_OPTS_MUTUAL_REQUIRED
in_data (input)  Application data whose checksum should be included in the authenticator. Specify NULL if no checksum is to be included.
in_creds (input)  Specifies the credentials for the service.
outbuf (output)  A pointer to an existing krb5_data structure to be filled. Returns the generated AP_REQ message.

Description

This routine formats a KRB_AP_REQ message into outbuf, with more complete options than krb5_mk_req.
The outbuf, ap_req_options, auth_context, and ccache arguments are used in the same fashion as for krb5_mk_req.
The in_creds argument is used to supply the credentials (ticket and session key) needed to form the request.
If in_creds->ticket has no data (length == 0), then an error is returned.
During a call to this routine, the structure elements in in_creds may be freed and reallocated. Hence all of the structure elements which are pointers should point to allocated memory, and there should be no other pointers aliased to the same memory, since it may be deallocated during this routine call.
If ap_req_options specifies AP_OPTS_USE_SUBKEY, then a subkey will be generated if need be by krb5_generate_subkey.
A copy of the authenticator will be stored in the auth_context, with the principal and checksum fields nulled out, unless an error is returned. (This is to prevent pointer-sharing problems; the caller should not need these fields anyway, since the caller supplied them.)
## Return Values

This routine returns one of the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Insufficient memory.</td>
</tr>
</tbody>
</table>
**krb5_mk_safe — Format a KRB_SAFE message**

**C Prototype**

```c
krb5_error_code krb5_mk_safe(
    krb5_context context,
    krb5_auth_context *auth_context,
    const krb5_data *userdata,
    krb5_data *outbuf,
    krb5_replay_data *outdata );
```

**Arguments**

- **context (input/output)** The context structure.
- **auth_context (input/output)** Authentication context. The auth_context->auth_context_flags select whether sequence numbers or timestamps should be used to identify the message. Valid flags are:
  - `KRB5_AUTH_CONTEXT_DO_TIME` — Use timestamps and replay cache.
  - `KRB5_AUTH_CONTEXT_RET_TIME` — Copy timestamp to *outdata.
  - `KRB5_AUTH_CONTEXT_DO_SEQUENCE` — Use sequence numbers.
  - `KRB5_AUTH_CONTEXT_RET_SEQUENCE` — Copy sequence numbers to *outdata.
- **userdata (input)** The user data in the message.
- **outbuf (output)** The formatted KRB_SAFE buffer.
- **outdata (input/output)** Contains the sequence numbers if `KRB5_AUTH_CONTEXT_RET_SEQUENCE` was specified in auth_context.

**Description**

This routine formats a KRB_SAFE message into outbuf.

The userdata argument is formatted as the user data in the message. Portions of auth_context specify the checksum type, the keyblock that might be used to seed the checksum, and full addresses (host and port) for the sender and receiver. The local_addr portion of auth_context is used to form the addresses used in the KRB_SAFE message. The remote_addr is optional; if the receiver's address is not known, it may be replaced by NULL. The local_addr argument, however, is mandatory.

If timestamps are to be used (that is, if KRB5_AUTH_CONTEXT_DO_TIME is set), an entry describing the message will be entered in the replay cache so that the caller may detect if this message is sent back by an attacker. If KRB5_AUTH_CONTEXT_DO_TIME is not set, the auth_context replay cache is not used.

If sequence numbers are to be used (if either KRB5_AUTH_CONTEXT_DO_SEQUENCE or KRB5_AUTH_CONTEXT_RET_SEQUENCE is set), then auth_context local sequence number will be placed in the protected message as its sequence number.

The outbuf buffer storage (outbuf->data) is allocated, and should be freed by the caller when finished.
Return Values

This routine returns one of the following KRB5 status codes:

0                    Successful completion.
krb5_os_localaddr — Return all protocol addresses of this host

C Prototype

```c
krb5_error_code krb5_os_localaddr(
    krb5_context context,
    krb5_address ***addr );
```

Arguments

- **context** (input) The context structure.
- **addr** (output) A pointer to an array of address pointers.

Description

This routine returns all of the protocol addresses of this host.

Compile-time configuration flags will indicate which protocol family addresses might be returned. The `*addr` argument is filled in to point to an array of address pointers, terminated by a NULL pointer. All the storage pointed to is allocated and should be freed by the caller with `krb5_free_addresses` when no longer needed.

Return Values

This routine returns the following KRB5 status code:

- **0** Successful completion.
- **ENOMEM** Insufficient memory.
**krb5_parse_name — Convert string principal name to protocol format**

**C Prototype**

```c
krb5_error_code krb5_parse_name(
    krb5_context           context,
    const char             *name,
    krb5_principal         *principal );
```

**Arguments**

- **context (input/output)**: The context structure.
- **name (input)**: Single string representation of a Kerberos principal name.
- **principal (output)**: Multipart principal format used in the protocols.

**Description**

This routine converts a single-string representation name of the principal name to the multi-part principal format used in the protocols.

A single-string representation of a Kerberos name consists of one or more principal name components, separated by slashes, optionally followed by the @ character and a realm name. If the realm name is not specified, the local realm is used.

The slash and @ characters can be quoted (included as part of a component rather than as a component separator or realm prefix) by preceding them with a backslash (\) character. Similarly, newline, tab, backspace, and NULL characters can be included in a component by using \n, \t, \b or \0, respectively.

The realm in a Kerberos name cannot contain the slash, colon, or NULL characters.

The *principal argument points to allocated storage that should be freed by the caller (using krb5_free_principal) after use.

**Return Values**

This routine returns one of the following KRB5 status codes:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
krb5_principal2salt — Convert a krb5_principal into its default salt

**C Prototype**

```
krb5_error_code krb5_principal2salt (  
    krb5_context context,  
    register krb5_const_principal pr,  
    krb5_data *ret );
```

**Arguments**

- **context (input/output)**: The context structure.
- **pr (input)**: The principal name.
- **ret (output)**: A pointer to the default salt for the given principal name.

**Description**

This routine converts a principal name into the default salt for that principal.

**Return Values**

This routine returns one of the following KRB5 status codes:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
krb5_principal_compare — Compare two principals

C Prototype

```c
krb5_boolean krb5_principal_compare(
    krb5_context              context,
    krb5_const_principal      princ1,
    krb5_const_principal      princ2 );
```

Arguments

- **context (input/output)**: The context structure.
- **princ1 (input)**: First principal name.
- **princ2 (input)**: Second principal name.

Description

This routine compares two principal names.

Return Values

This routine returns one of the following KRB5 status codes:

- **TRUE**: Principals are the same.
- **FALSE**: Principals are different.
**krb5_prompter_posix — Prompt the user for the Kerberos password**

**C Prototype**

```c
krb5_error_code krb5_prompter_posix (  
    krb5_context            context,  
    void                    *data,  
    const char              *name,  
    const char              *banner,  
    int                     num_prompts,  
    krb5_prompt             prompts[] );
```

**Arguments**

- **context** (input/output) The context structure.
- **data** (input) [Not used].
- **name** (input) Name to output during prompt.
- **banner** (input) Banner to output during prompt.
- **num_prompts** (input) The number of prompts passed in prompts.
- **prompts** (input/output) A structure containing output prompts and replies.

**Description**

This routine prompts the user for the Kerberos password associated with the given principal name, and sets the reply field of the prompts argument to the password input. The hidden flag in the prompts structure controls whether the password input is echoed back to the terminal.

**Return Values**

This routine returns one of the following KRB5 status code:

- **0** Successful completion.
krb5_rd_cred — Read a KRB_CRED message

C Prototype

```c
krb5_error_code krb5_rd_cred (  
    krb5_context            context,  
    krb5_auth_context       auth_context,  
    krb5_data               *pcreddata,  
    krb5_creds              ***pppcreds,  
    krb5_replay_data        *outdata );
```

Arguments

- context (input/output) The context structure.
- auth_context (input/output) A per-connection context.
- pcreddata (input) The KRB_CRED message.
- pppcreds (output) The array of forwarded credentials.
- outdata (output) The replay data information (the nonce).

Description

This routine reads a KRB_CRED message, validates it, and outputs the nonce and an array of the forwarded credentials.

Return Values

This routine returns one of the following KRB5 status codes:

- 0: Successful completion.
- KRB5_RC_REQUIRED: Message replay detection requires rcache parameter.
- KRB5_KRB_AP_ERR_SKEW: Clock skew too great.
- KRB5_KRB_AP_ERR_BADORDER: Message out of order.
- ENOMEM: Insufficient memory.
krb5_rd_error — Read an error protocol message

C Prototype

```c
krb5_error_code krb5_rd_error (  
    krb5_context    context,  
    const krb5_data *enc_errbuf,  
    krb5_error      **dec_error );
```

Arguments

context (input) The context structure.
enc_errbuf (input) A pointer to the error protocol message.
dec_error (output) A pointer to allocated storage containing the error message.

Description

Parses an error protocol message from `enc_errbuf` and fills in `*dec_error` with a pointer to allocated storage containing the error message. The caller is responsible for freeing this structure by using `krb5_free_error`.

Return Values

This routine returns one of the following KRB5 status code:

```
0       Successful completion.
```
**krb5_rd_priv — Parse a KRB_PRIV message**

**C Prototype**

```c
krb5_error_code krb5_rd_priv(
    krb5_context           context,
    krb5_auth_context      auth_context,
    const krb5_data        *inbuf,
    krb5_data              *outbuf,
    krb5_data              *outdata);
```

**Arguments**

- **context** (input/output): The context structure.
- **auth_context** (input/output): Authentication context.
- **inbuf** (input): The KRB_PRIV message to be parsed.
- **outbuf** (output): The data parsed from the KRB_PRIV message.
- **outdata** (input/output): Contains the sequence numbers if KRB5_AUTH_CONTEXT_RET_SEQUENCE was specified in **auth_context**.

**Description**

This routine parses a KRB_PRIV message from **inbuf**, placing the data in **outbuf** after decrypting it. It behaves similarly to **krb5_rd_safe**, but the message is decrypted rather than integrity checked.

The **inbuf**, **auth_context**, **outdata** and **outbuf** arguments function as in **krb5_rd_safe**.

The **remote_addr** part of the **auth_context** as set by **krb5_auth_con_setaddr** is mandatory; it specifies the address of the sender. If the address of the sender in the message does not match the **remote_addr**, the error KRB5KRB_AP_ERR_BADADDR will be returned.

If **local_addr** portion of the **auth_context** is **nonNULL**, then the address of the receiver in the message must match it. If it is **NULL**, the receiver address in the message will be checked against the list of local addresses as returned by **krb5_os_localaddr**.

The **keyblock** portion of **auth_context** specifies the key to be used for decryption of the message. If the **i_vector** element is **nonNULL**, it is used as an initialization vector for the decryption (if the encryption type of the message supports initialization vectors) and its contents are replaced with the last block of encrypted data in the message.

The **auth_context** flags specify whether timestamps (KR5_AUTH_CONTEXT_DO_TIME) and sequence numbers (KR5_AUTH_CONTEXT_DO_SEQUENCE) are to be used.

**Return Values**

This routine returns one of the following KRB5 status codes:

- **0**: Successful completion.
**krb5_rd_rep — Parse and decrypt an AP_REP message**

**C Prototype**

```c
krb5_error_code krb5_rd_rep(
    krb5_context           context,
    krb5_auth_context      auth_context,
    const krb5_data        *inbuf,
    krb5_ap_rep_enc_part   **repl );
```

**Arguments**

- **context (input/output)**: The context structure.
- **auth_context (input/output)**: Authentication context.
- **inbuf (input)**: The AP_REP message to parse and decrypt.
- **repl (output)**: The parsed message.

**Description**

This routine parses and decrypts an AP_REP message from *inbuf*, filling in *repl* with a pointer to allocated storage containing the values from the message. The caller is responsible for freeing this structure with `krb5_free_ap_rep_enc_part`.

The keyblock stored in `auth_context` is used to decrypt the message after establishing any key preprocessing with `krb5_process_key`.

**Return Values**

This routine returns one of the following KRB5 status codes:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
krb5_rd_req — Parse a KRB_AP_REQ message

C Prototype

```c
krb5_error_code krb5_rd_req(
    krb5_context              context,
    krb5_auth_context         *auth_context,
    const krb5_data           *inbuf,
    krb5_const_principal      server,
    krb5_keytab               keytab,
    krb5_flags                *ap_req_options,
    krb5_ticket               **ticket );
```

Arguments

- **context**: (input/output) The context structure.
- **auth_context**: (input/output) Authentication context. A new authentication context will be returned if NULL is specified.
- **inbuf**: (input) Contains the KRB_AP_REQ message to be parsed.
- **server**: (input) Specifies the expected server's principal name for the ticket.
- **keytab**: (input) Specifies a keytab containing a decryption key. If NULL, `krb5_kt_default` will be used to find the default keytab and the key taken from there.
- **ap_req_options**: (input/output) If nonNULL on input, this field will be set to contain the application request flags on output.
- **ticket**: (output) Returns the ticket from the AP_REQ message. The caller is responsible for deallocating this space by using `krb5_free_ticket`. If no ticket is desired, specify NULL.

Description

This routine parses a KRB_AP_REQ message, returning its contents. Upon successful return, if ticket is nonNULL, `*ticket` will be modified to point to allocated storage containing the ticket information.

If **auth_context** is NULL, one will be generated and freed internally by the function.

The **server** argument specifies the expected server's name for the ticket.

If **server** is NULL, then any server name will be accepted if the appropriate key can be found, and the caller should verify that the server principal matches some trust criterion.

If **server** is not NULL, and a replay detection cache has not been established with **auth_context**, one will be generated.

If a keyblock is present in the **auth_context**, it will be used to decrypt the ticket request and the keyblock freed with `krb5_free_keyblock`. This is useful for user-to-user authentication.

If no keyblock is specified, the **keytab** is consulted for an entry matching the requested keytype, server, and version number and used instead.
The authenticator in the request is decrypted and stored in `auth_context`. The client specified in the decrypted authenticator is compared to the client specified in the decoded ticket to ensure that the compare was performed.

If the `remote_addr` portion of the `auth_context` is set, then this routine checks if the request came from the right client.

The replay cache is checked to see if the ticket and authenticator have been seen and, if so, returns an error. If not, the ticket and authenticator are entered into the cache.

Various other checks are made of the decoded data, including cross-realm policy, clockskew, and ticket validation times.

The keyblock, subkey, and sequence number of the request are all stored in the `auth_context` for future use.

If the request has the AP_OPTS_MUTUAL_REQUIRED bit set, the local sequence number, which is stored in the `auth_context`, is XORed with the remote sequence number in the request.

**Return Values**

This routine returns one of the following KRB5 status codes:

- **0**: Successful completion.
- **KRB5_KRB_AP_ERR_BADADDR**: Invalid address.
**krb5_rd_safe — Parse a KRB_SAFE message**

### C Prototype

```c
krb5_error_code krb5_rd_safe(
    krb5_context context,
    krb5_auth_context *auth_context,
    const krb5_data *inbuf,
    krb5_data *outbuf,
    krb5_replay_data *outdata );
```

### Arguments
- **context** (input/output): The context structure.
- **auth_context** (input/output): Authentication context.
- **inbuf** (input): The KRB_SAFE message to be parsed.
- **outbuf** (output): The data parsed from the KRB_SAFE message.
- **outdata** (input/output): Contains the sequence numbers if `KRB5_AUTH_CONTEXT_RET_SEQUENCE` was specified in `auth_context`.

### Description
This routine parses a KRB_SAFE message from `inbuf`, placing the data in `outbuf` after verifying its integrity.

The keyblock used for verifying the integrity of the message is taken from the `auth_context local_subkey`, `remote_subkey`, or `keyblock`. The keyblock is chosen in the preceding order by the first one that is not NULL.

The `remote_addr` and `local_addr` portions of the `*auth_context` specify the full addresses (host and port) of the sender and receiver, and must be of type ADDRTYPE_ADDRPORT.

The `remote_addr` argument is mandatory; it specifies the address of the sender. If the address of the sender in the message does not match `remote_addr`, the error `KRB5KRB_AP_ERR_BADADDR` will be returned.

If `local_addr` is nonNULL, then the address of the receiver in the message much match it. If it is NULL, the receiver address in the message will be checked against the list of local addresses as returned by `krb5_os_localaddr`. If the check fails, `KRB5KRB_AP_ERR_BADADDR` is returned.

The `outbuf` buffer storage (`outbuf->data`) is allocated storage which the caller should free when it is no longer needed.

If `auth_context_flags` portion of `auth_context` indicates that sequence numbers are to be used (if `KRB5_AUTH_CONTEXT_DOSEQUENCE` is set in it), the `remote_seq_number` portion of `auth_context` is compared to the sequence number for the message, and `KRB5_KRB_AP_ERR_BADORDER` is returned if it does not match. Otherwise, the sequence number is not used.

If timestamps are to be used (if `KRB5_AUTH_CONTEXT_DO_TIME` is set in `auth_context`), then two additional checks are performed:
The timestamp in the message must be within the permitted clock skew (which is usually five minutes), or KRB5KRB_AP_ERR_SKEW is returned.

- The message must not be a replayed message, according to rcache.

**Return Values**

This routine returns one of the following KRB5 status codes:

- 0  
  Successful completion.
krb5_read_password — Read a password from the keyboard

C Prototype

```c
krb5_error_code krb5_read_password(
    krb5_context     context,
    const char       *prompt,
    const char       *prompt2,
    char             *return_pwd,
    unsigned int     *size_return );
```

Arguments

- `context` (input): The context structure.
- `prompt` (input): First user prompt when reading password.
- `prompt2` (input): Second user prompt, or NULL to read the password only once.
- `return_pwd` (output): The returned password.
- `size_return` (input/output): On input, the maximum size of the password to be returned. On output, the total number of bytes returned in `return_pwd`.

Description

This routine reads a password from the keyboard. The first `*size_return` bytes of the password entered are returned in `return_pwd`. If fewer than `*size_return` bytes are typed as a password, the remainder of `return_pwd` is zeroed. Upon success, the total number of bytes filled in is stored in `*size_return`.

The `prompt` argument is used as the prompt for the first reading of a password. It is printed to the terminal, and then a password is read from the keyboard. No newline or spaces are emitted between the prompt and the cursor, unless the newline/space is included in the prompt.

If `prompt2` is a NULL pointer, then the password is read once.

If `prompt2` is set, then it is used as a prompt to read another password in the same manner as described for `prompt`. After the second password is read, the two passwords are compared, and an error is returned if they are not identical.

Echoing is turned off when the password is read.

Return Values

This routine returns one of the following KRB5 status codes:

- 0: Successful completion.
- Error in reading or verifying the password.
**krb5_realm_compare — Compare the realms of two principals**

**C Prototype**

```c
krb5_boolean krb5_realm_compare (  
    krb5_context            context,  
    krb5_const_principal    princ1,  
    krb5_const_principal    princ2 );
```

**Arguments**

- context (input/output): The context structure.
- princ1 (input): First principal name.
- princ2 (input): Second principal name.

**Description**

Compares two realms. If the realms of the two principals are the same, return TRUE, else return FALSE.

**Return Values**

This routine returns one of the following KRB5 status codes:

- **True.** The realms are the same.
- **False.** The realms are the different.
krb5_recvauth — Receive authenticated message

C Prototype

```c
krb5_error_code krb5_recvauth(
    krb5_context           context,
    krb5_auth_context      *auth_context,
    krb5_pointer           fd,
    char                   *appl_version,
    krb5_principal         server,
    krb5_int32             flags,
    krb5_keytab            keytab,
    krb5_ticket            **ticket );
```

Arguments

- **context** (input/output) The context structure.
- **auth_context** (input/output) Authentication context.
- **fd** (input) A pointer to a file descriptor describing the network socket.
- **appl_version** (input) A string describing the application protocol version that the client is expecting to use for this exchange. If the client is using a different application protocol, an error will be returned, and the authentication exchange will be aborted.
- **server** (input) If server is nonNULL, then `krb5_recvauth` verifies that the server principal requested by the client matches server. If not, an error will be returned, and the authentication exchange will be aborted.
- **flags** (input) The flags argument allows the caller to modify the behavior of `krb5_recvauth`. For nonlibrary callers, flags should be 0.
- **keytab** (input) Specifies a keytab containing a decryption key.
- **ticket** (output) Ticket is optional and is only filled in if nonNULL. It is filled with the data from the ticket sent by the client, and should be freed with `krb5_free_ticket` when it is no longer needed.

Description

This routine provides a convenient means for client and server programs to send authenticated messages to one another through network connections. The `krb5_sendauth` routine is the matching routine to `krb5_recvauth` for the server. The `krb5_recvauth` routine will engage in an authentication dialog with the client program running `krb5_sendauth` to authenticate the client to the server. In addition, if requested by the client, `krb5_recvauth` will provide mutual authentication to prove to the client that the server represented by `krb5_recvauth` is legitimate.

The `fd` argument is a pointer to the network connection. As in `krb5_sendauth`, in the MIT UNIX and OpenVMS implementations, `fd` is a pointer to a file descriptor.

The arguments `server`, `auth_context`, and `keytab` are used by `krb5_rd_req` to obtain the server's private key.
If `server` is `nonNULL`, the principal component of it is used to determine the replay cache to use. Otherwise, `krb5_recvauth` will use a default replay cache.

**Return Values**

This routine returns the following KRB5 status code:

```
0                     Successful completion.
```
krb5_recvauth_version — Receive authenticated message with version information

C Prototype

```c
krb5_error_code krb5_recvauth_version (  
    krb5_context            context,  
    krb5_auth_context       *auth_context,  
    krb5_pointer            fd,  
    krb5_principal          server,  
    krb5_int32              flags,  
    krb5_keytab             keytab,  
    krb5_ticket             **ticket,  
    krb5_data               *version );
```

Arguments

context (input/output)  The context structure.
auth_context (input)    The Kerberos authentication context.
fd (input)              The socket from which to read the client responses.
server (input)          If server is nonNULL, then krb5_recvauth_version verifies that the server principal requested by the client matches server. If it is NULL, an error is returned and the authentication exchange is aborted.
flags (input)           Allows the caller to modify the behavior of krb5_recvauth_version. For nonlibrary callers, flags should be 0.
keytab (input)          A Kerberos keytab, containing a decryption key.
ticket (output)         Optional argument that is filled in only if nonNULL. It is filled with the data from the ticket sent by the client, and should be freed with krb5_free_ticket when it is no longer needed.
version (output)        A pointer to the application version string.

Description

This routine provides a convenient means for client and server programs to send authenticated messages to one another through network connections. (The k5b5_sendauth routine is the matching routine to krb5_recvauth_version for the server.)

The krb5_recvauth_version routine engages in an authentication dialog with the client program running k5b5_sendauth to authenticate the client to the server. In addition, if requested by the client, krb5_recvauth_version provides mutual authentication to prove to the client that the server represented by krb5_recvauth_version is legitimate.

The fd argument is a pointer to the network connection. As in k5b5_sendauth, in the MIT UNIX and OpenVMS implementations, fd is a pointer to a file descriptor.

The arguments server, auth_context, and keytab are used by krb5_rd_req to obtain the server’s private key.
If `server` is non-NULL, the principal component of it is used to determine the replay cache to use. Otherwise, `krb5_recvauth_version` uses a default replay cache.

**Return Values**

This routine returns one of the following KRB5 status codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>KRB5_SENDAUTH_BADAUTHVERS</td>
<td>Bad sendauth version was sent.</td>
</tr>
<tr>
<td>KRB5_SENDAUTH_BADAPPLVERS</td>
<td>Bad application version was sent (via sendauth).</td>
</tr>
</tbody>
</table>


**krb5_salttype_to_string — Convert a salttype (krb5_int32) to a string**

**C Prototype**

```c
krb5_error_code krb5_salttype_to_string (        
    krb5_int32      salttype,               
    char            *buffer,               
    size_t          buflen );              
```

**Arguments**

- `salttype` (input) The salttype to convert.
- `buffer` (output) A pointer to the buffer to receive the converted string.
- `buflen` (input) The length of buffer.

**Description**

This routine converts a salttype (krb5_int32) into a string.

**Return Values**

This routine returns the following KRB5 status codes:

- **0** Successful completion.
- **EINVAL** Invalid parameter.
- **ENOMEM** Insufficient memory (buffer length less than output size).
KRB5 (Kerberos V5) Application Programming Interface

krb5_sendauth — Send authenticated message

C Prototype

```c
krb5_error_code krb5_sendauth(
    krb5_context context,
    krb5_auth_context *auth_context,
    krb5_pointer fd,
    char *appl_version,
    krb5_principal client,
    krb5_principal server,
    krb5_flags ap_req_options,
    krb5_data *in_data,
    krb5_creds *in_creds,
    krb5_ccache ccache,
    krb5_error **error,
    krb5_ap_rep_enc_part **rep_result,
    krb5_creds **out_creds );
```

Arguments

- context (input/output) The context structure.
- auth_context (input/output) Authentication context.
- fd (input) A pointer to a file descriptor describing the network socket.
- appl_version (input) A string describing the application protocol version that the client is expecting to use for this exchange. If the server is using a different application protocol, an error will be returned.
- client (input) The Kerberos principal for the client. Ignored if in_creds is nonNULL.
- server (input) The Kerberos principal for the server. Ignored if in_creds is nonNULL.
- ap_req_options (input) Specifies the KRB_AP_REQ flags that should be passed to krb5_mk_req. Valid flags are:
  - AP_OPTS_USE_SESSION_KEY
  - AP_OPTS_MUTUAL_REQUIRED
  - AP_OPTS_USE_SUBKEY
- in_data (input) The data to be sent to the server.
- in_creds (input) Input credentials, or NULL.
- ccache (input/output) The credentials cache.
- error (output) If nonNULL, contains the error packet returned from the server. This error should be freed with krb5_free_error.
- rep_result (output) If nonNULL, contains the result of the mutual authentication exchange. The *rep_result argument should be freed with krb5_free_ap_rep_enc_part when the caller is done with it.
- out_creds (output) If nonNULL, the retrieved credentials.
Description

This routine provides a convenient means for client and server programs to send authenticated messages to one another through network connections. The krb5_sendauth routine sends an authenticated ticket from the client program to the server program using the network connection specified by fd. In the MIT UNIX and OpenVMS implementations, fd should be a pointer to a file descriptor describing the network socket.

The arguments client and server specify the Kerberos principals for the client and the server. They are ignored if in_creds is nonNULL. Otherwise, server must be nonNULL, but client may be NULL, in which case the client principal used is the one in the credential cache's default principal.

The ap_req_options argument specifies the options that should be passed to krb5_mk_req. If ap_req_options specifies MUTUAL_REQUIRED, then krb5_sendauth will perform a mutual authentication exchange, and if rep_result is nonNULL, it will be filled in with the result of the mutual authentication exchange; the caller should free *rep_result with krb5_free_ap_rep_enc_part when done with it.

If in_creds is nonNULL, then in_creds->client and in_creds->server must be filled in, and either the other structure fields should be filled in with valid credentials, or in_creds->ticket.length should be zero. If in_creds->ticket.length is nonzero, then in_creds will be used as-is as the credentials to send to the server, and ccache is ignored; otherwise, ccache is used as described later, and out_creds, if not NULL, is filled in with the retrieved credentials.

The ccache argument specifies the credential cache to use when one is needed (that is, when in_creds is NULL or in_creds->ticket.length is zero). When a credential cache is not needed, ccache is ignored. When a credential cache is needed and ccache is NULL, the default credential cache is used. Note that if the credential cache is needed and does not contain the needed credentials, they will be retrieved from the KDC and stored in the credential cache.

If mutual authentication is used and rep_result is nonNULL, the sequence number for the server is available to the caller in *rep_result->seq_number. (If mutual authentication is not used, there is no way to negotiate a sequence number for the server.)

If an error occurs during the authenticated ticket exchange and error is nonNULL, the error packet (if any) that was sent from the server will be placed in it. This error should be freed with krb5_free_error.

Return Values

This routine returns the following KRB5 status code:

0 Successful completion.
krb5_set_default_realm — Sets the default realm

C Prototype

```c
krb5_error_code krb5_set_default_realm(
    krb5_context context,
    char *realm );
```

Arguments

- context (input) The context structure.
- realm (output) The default realm to be set. If realm is NULL, then the operating system default value will be used.

Description

This routine sets the default realm to be used if no user-specified realm is available (for example, to interpret a user-typed principal name with the realm omitted for convenience).

Return Values

This routine returns the following KRB5 status code:

- 0 Successful completion.
- ENOMEM Insufficient memory.
**krb5_set_default_tgs_enctypes — Set default TGS encryption types**

**C Prototype**

```c
krb5_error_code krb5_set_default_tgs_enctypes (  
    krb5_context context,  
    const krb5_enctype *ktypes);
```

**Arguments**

- **context (input/output)** The context structure.
- **ktypes (input)** The encryption type(s) to set as the default.

**Description**

This routine sets the default Ticket Granting Service encryption types for the given Kerberos context.

**Return Values**

This routine returns the following KRB5 status codes:

- 0: Successful completion.
- ENOMEM: Insufficient memory.
- KRB5_PROG_ETYPE_NOSUPP: Program lacks support for encryption type.
krb5_set_password — Implements set password per RFC 3244

C Prototype

```c
krb5_error_code krb5_set_password (  
    krb5_context context,  
    krb5_creds *creds,  
    char *newpw,  
    krb5_principal change_password_for,  
    int *result_code,  
    krb5_data *result_code_string,  
    krb5_data *result_string);
```

Arguments

- **context (input/output)**: The context structure.
- **creds (input)**: The Kerberos credentials.
- **newpw (input)**: The new password to be set.
- **change_password_for (input)**: The principal for which the password change is to be performed. If NULL, then the change is to be done on the current principal. If nonNULL, the change is performed on the principal name passed in change_password_for.
- **result_code (output)**: A pointer to the result code returned from the remote system.
- **result_code_string (output)**: A pointer to the result code translated into a readable message.
- **result_string (output)**: A pointer to the result data returned from the remote system.

Description

This routine allows a new password to be set in a manner that is interoperable with Windows implementations, per RFC 3244.

Return Values

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **INVALID_SOCKET**: Invalid socket.
- **ECONNREFUSED**: Connection refused.
- **EHOSTUNREACH**: Host unreachable.
- **ECONNABORTED**: Connection aborted.
- **ETIMEDOUT**: Connection timed out.
- **ENOMEM**: Insufficient memory.
krb5_set_password_using_ccache — Implements RFC 3244 set password using credentials cache

C Prototype

```c
krb5_error_code krb5_set_password_using_ccache(
    krb5_context    context,
    krb5_ccache     ccache,
    char            *newpw,
    krb5_principal  change_password_for,
    int             *result_code,
    krb5_data       *result_code_string,
    krb5_data       *result_string );
```

Arguments

- context (input/output) The context structure.
- ccache (input) The Kerberos credentials cache.
- newpw (input) The new password to be set.
- change_password_for (input) The principal for which the password change is to be performed. If NULL, the change is to be done on the current principal. If nonNULL, the change is preformed on the principal name passed in `change_password_for`.
- result_code (output) A pointer to the result code returned from the remote system.
- result_code_string (output) A pointer to the result code translated into a readable message.
- result_string (output) A pointer to the result data returned from the remote system.

Description

This routine allows a new password to be set in a manner that is interoperable with Windows implementations per RFC 3244. This routine uses the credentials cache instead of explicitly passed credentials (which are used in `krb5_set_password`).

Return Values

This routine returns the following KRB5 status codes:

- 0 Successful completion.
- INVALID_SOCKET Invalid socket.
- ECONNREFUSED Connection refused.
- EHOSTUNREACH Host unreachable.
- ECONNABORTED Connection aborted.
- ETIMEDOUT Connection timed out.
- ENOMEM Insufficient memory.
**C Prototype**

```c
krb5_error_code krb5_set_principal_realm ( 
    krb5_context context, 
    krbt5_principal principal, 
    const char *realm );
```

**Arguments**

- **context (input/output)**: The context structure.
- **principal (input)**: The Kerberos principal name.
- **realm (input)**: The new realm to which the context should be set.

**Description**

This routine sets the realm in the current context to `realm`.

**Return Values**

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **EINVAL**: Invalid parameter.
- **ENOMEM**: Insufficient memory.
krb5_set_real_time — Set time offset field in context structure

C Prototype

```c
krb5_error_code krb5_set_real_time (
    krb5_context            context,
    krb5_int32              seconds,
    krb5_int32             microseconds );
```

Arguments

- context (input/output) The context structure.
- seconds (input) The number of seconds to set in the time_offset field in the context.
- microseconds (input) The number of microseconds to set in the usec_offset field in the context.

Description

This routine takes the “real time” as input, and sets the time offset fields in the Kerberos context structure so that the krb5 time routines will return the correct time, as corrected by the difference between the system time and the “real time” as passed to this routine.

Return Values

This routine returns the following KRB5 status codes:

```
0             Successful completion.
```
krb5_sname_to_principal — Generate a full principal name from a service name

C Prototype

```c
krb5_error_code krb5_sname_to_principal(
    krb5_context context,
    const char *hostname,
    const char *sname,
    krb5_int32 type,
    krb5_principal *ret_princ );
```

Arguments

- **context (input)**: The context structure.
- **hostname (input)**: The host name, or NULL to use the local host.
- **sname (input)**: The service name.
- **type (input)**: A principal type. The type argument controls how `krb5_sname_to_principal` generates the principal name, `ret_princ`, for the named service, `sname`. Valid values are:
  - `KRB5_NT_SRV_HST` — The hostname will be canonicalized (a fully qualified lowercase hostname using the primary name and the domain name), before `ret_princ` is generated in the form `sname/hostname@LOCAL.REALM`. Most applications should use `KRB5_NT_SRV_HST`.
  - `KRB5_NT_UNKNOWN` — While the generated principal name will have the form `sname/hostname@LOCAL.REALM`, the hostname will not be canonicalized first. It will appear exactly as it was passed in `hostname`.
- **ret_princ (output)**: The returned full principal name.

Description

This routine generates a full principal name to be used when authenticating with the named service on the host, given a hostname `hostname` and a generic service name `sname`. The full principal name is returned in `ret_princ`.

The realm of the principal is determined internally by calling `krb5_get_host_realm`.

The caller should release the storage in `ret_princ` by calling `krb5_free_principal` when it is finished with the principal.

Return Values

This routine returns the following KRB5 status code:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.
**krb5_string_to_cksumtype — Convert a string to a checksum type**

**C Prototype**

```c
krb5_error_code krb5_string_to_cksumtype(
    char *string,
    krb5_cksumtype *cksumtypep);
```

**Arguments**

- **string (input)**: A pointer to the string value to convert to a checksum type.
- **cksumtypep (output)**: A pointer to the checksum type.

**Description**

This routine converts a character string into a Kerberos checksum type.

**Return Values**

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **EINVAL**: Invalid parameter.
krb5_string_to_deltat — Convert a string to a delta time value

C Prototype

```c
krb5_error_code krb5_string_to_deltat (  
    char                    *string,  
    krb5_deltat             *deltatp );
```

Arguments

- **string (input)**
  A pointer to the string to convert to a delta time.
- **deltatp (output)**
  A pointer to the delta time.

Description

This routine converts a string to a delta time value for use in other Kerberos routines.

Return Values

This routine returns the following KRB5 status codes:

- **0**
  Successful completion.
- **EINVAL**
  Invalid parameter.
krb5_string_to_enctype — Convert a string to an encryption type

C Prototype

```c
krb5_error_code krb5_string_to_enctype(
    char *string,
    krb5_enctype *enctypep
);
```

Arguments

- **string (input)**: A pointer to the string to convert to an encryption type.
- **deltatp (output)**: A pointer to the encryption type.

Description

This routine converts a string to a Kerberos encryption type.

Return Values

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **EINVAL**: Invalid parameter.
krb5_string_to_salttype — Convert a string to a salt type

C Prototype

```c
krb5_error_code krb5_string_to_salttype (
    char                    *string,
    krb5_int32              *salttypep);
```

Arguments

- `string` (input): A pointer to the string to convert to a salt type.
- `salttypep` (output): A pointer to the salt type.

Description

This routine converts a string to a Kerberos salt type.

Return Values

This routine returns the following KRB5 status codes:

- 0: Successful completion.
- EINVAL: Invalid parameter.
krb5_string_to_timestamp — Convert a string to a timestamp

C Prototype

```c
krb5_error_code krb5_string_to_timestamp ( char *string,
    krb5_timestamp *timestampp );
```

Arguments

- `string` (input) A pointer to the string to convert to a timestamp.
- `timestampp` (output) A pointer to the timestamp.

Description

This routine converts a string to a Kerberos timestamp.

Return Values

This routine returns the following KRB5 status codes:

- 0 Successful completion.
- EINVAL Invalid parameter.
**krb5_timestamp_to_sfstring — Convert a timestamp to a string**

**C Prototype**

```c
krb5_error_code krb5_timestamp_to_sfstring (  
    krb5_timestamp          timestamp,  
    char                    *buffer,  
    size_t                  buflen,  
    char                    *pad );
```

**Arguments**

- **timestamp (input)**: The timestamp to convert.
- **buffer (output)**: The buffer to hold the converted timestamp.
- **buflen (input)**: The length of buffer.
- **pad (input)**: If the converted timestamp does not fill `buffer`, an optional value used to pad the rest of `buffer`.

**Description**

This routine converts a Kerberos timestamp to a string.

**Return Values**

This routine returns the following KRB5 status codes:

- **0**: Successful completion.
- **ENOMEM**: Insufficient memory.


**krb5_timestamp_to_string — Convert a timestamp to a string**

**C Prototype**

```c
krb5_error_code krb5_timestamp_to_string (  
    krb5_timestamp          timestamp,  
    char                    *buffer,  
    size_t                  buflen );
```

**Arguments**

- `timestamp` (input) The timestamp to convert.
- `buffer` (output) The buffer to hold the converted timestamp.
- `buflen` (input) The length of buffer.

**Description**

This routine converts a Kerberos timestamp to a string. It returns the string in the locale's appropriate date and time representation.

**Return Values**

This routine returns the following KRB5 status codes:

- `0` Successful completion.
- `ENOMEM` Insufficient memory.
krb5_unparse_name — Convert protocol format principal name to string format

C Prototype

```c
krb5_error_code krb5_unparse_name ( 
    krb5_context context, 
    krb5_const_principal principal, 
    char **name );
```

Arguments

- context (input/output) The context structure.
- principal (input) Multipart principal format used in the protocols.
- name (output) Single string representation of a Kerberos principal name.

Description

This routine converts the multipart principal name principal from the format used in the protocols to a single-string representation of the name. The resulting single-string representation will use the format and quoting conventions described for `krb_parse_name`.

The `name` argument points to allocated storage and should be freed by the caller when finished.

Return Values

This routine returns one of the following KRB5 status codes:

- 0 Successful completion.
- ENOMEM Insufficient memory.
krb5_unparse_name_ext — Convert multiple protocol format principal names to string format

C Prototype

krb5_error_code krb5_unparse_name_ext(  
    krb5_context context,  
    krb5_const_principal principal,  
    char **name,  
    int *size );

Arguments

context (input/output)  The context structure.
principal (input)  Multipart principal format used in the protocols.
name (output)  Single string representation of a Kerberos principal name.
size (output)  Size of the unparsed name buffer.

Description

This routine is designed for applications which must unparse a large number of principals, and are concerned about the speed impact of needing to do a lot of memory allocations and deallocations. It functions similarly to krb5_unparse_name except if *name is nonNULL, in which case, it is assumed to contain an allocated buffer of size *size and this buffer will be resized with realloc to hold the unparsed name. Note that in this case, *size must not be NULL.

The *name argument points to allocated storage and should be freed by the caller when finished.

Return Values

This routine returns the following KRB5 status code:

    0  Successful completion.
    ENOMEM  Insufficient memory.
**KRB5 (Kerberos V5) Application Programming Interface**

**krb5_us_timeofday — Retrieves the system time of day (in seconds and microseconds)**

**C Prototype**

```c
krb5_error_code krb5_us_timeofday(
    krb5_context     context,
    krb5_int32       *seconds,
    krb5_int32       *microseconds );
```

**Arguments**

- **context (input)**: The context structure.
- **seconds (output)**: The system time of day, in seconds, since the local system’s epoch.
- **microseconds (output)**: The microseconds portion of the system time of day.

**Description**

This routine retrieves the system time of day, in seconds, since the local system’s epoch. The seconds portion is returned in *seconds, the microseconds portion in *microseconds.

**Return Values**

This routine returns the following KRB5 status code:

- 0: Successful completion.


**krb5_verify_init_creds — Verify initial credentials**

**C Prototype**

```c
krb5_error_code krb5_verify_init_creds (  
    krb5_context                    context,  
    krb5_creds                      *creds,  
    krb5_principal                  server_arg,  
    krb5_keytab                     keytab_arg,  
    krb5_ccache                     *ccache_arg,  
    krb5_verify_init_creds_opt      *options );
```

**Arguments**

- `context` (input/output) The context structure.
- `creds` (input) A pointer to the initial credentials.
- `server_arg` (input) The server principal.
- `keytab_arg` (input) The keytab entry.
- `ccache_arg` (input/output) A pointer to the credentials cache.
- `options` (input) A pointer to a structure containing flags and options.

**Description**

This routine verifies the set of initial credentials, and stores them in the credentials cache.

**Return Values**

This routine returns the following KRB5 status code:

- `0` Successful completion.
krb5_verify_init_creds_opt_init — Initialize krb5_verify_init_creds_opt structure

C Prototype

```c
void krb5_verify_init_creds_opt_init ( krb5_verify_init_creds_opt *opt );
```

Arguments

- **opt (output)** A pointer to the options field.

Description

This routine initializes the flags field in the `krb5_verify_init_creds_opt` structure.

Return Values

None.
krb5_verify_init_creds_opt_set_ap_req_nofail — Initialize the
ap_req_nofail field in krb5_verify_init_creds_opt

C Prototype

```c
void krb5_verify_init_creds_opt_set_ap_req_nofail (
    krb5_verify_init_creds_opt *opt,
    int ap_req_nofail );
```

Arguments

- `opt` (output) A pointer to the options field.
- `ap_req_nofail` (input) The value to set for the `ap_req_nofail` field in `opt`.

Description

This routine initializes the `ap_req_nofail` field in `krb5_verify_init_creds_opt` to `ap_req_nofail`, and sets the appropriate flag.

Return Values

None.
KRB5 (Kerberos V5) Application Programming Interface

`krb5_verify_init_creds_opt_set_ap_req_nofail` — Initialize the `ap_req_nofail` field in `krb5_verify_init_creds_opt`
A Open Source Notices

A.1 Acknowledgments

The Kerberos model is based in part on Needham and Schroeder's trusted third-party authentication protocol and on modifications suggested by Denning and Sacco. The original design and implementation of Kerberos Versions 1 through 4 was the work of Steve Miller of the former Digital Equipment Corporation (now Hewlett-Packard Company) 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 4 is publicly available, and has seen wide use across the Internet.

Version 5 (described in this document) has evolved from Version 4 based on new requirements and desires for features not available in Version 4.

A.2 Kerberos Copyright Notice

Copyright Notice, Kerberos © 1986-2001 by the Massachusetts Institute of Technology.

Export of software employing encryption from the United States of America may require a specific license from the United States Government.

It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Furthermore if you modify this software you must label your software as modified software and not distribute it in such a fashion that it might be confused with the original MIT software. MIT makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

A.3 OpenVision Technologies Copyright Notice

A.4 University of California Copyright Notice

Copyright Notice, University of California at Berkeley © 1983 Regents of the University of California.

MIT Kerberos includes documentation and software developed at the University of California at Berkeley, which includes this copyright notice:

All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
• All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes software developed by the University of California, Berkeley and its contributors."

• Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notices and this permission notice are preserved on all copies. Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one. Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.
Glossary

A-Z

**authentication** Verification of 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, the objects the client is allowed to access, and the type of access allowed.

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

**client** A process that uses a network service on behalf of a user. In some cases a server may itself be a client of some other server. (For example, a print server may be a client of a file server.)

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

**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** 1. In ancient mythology, the three-headed dog guarding Hades. 2. The name given to Project Athena's authentication service, the protocol used by that service, or the code used to implement the authentication service.

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

**principal** A uniquely named client or server instance that participates in a network communication.

**principal identifier** The name used to uniquely identify each different principal.

**realm** The administrative domain that encompasses Kerberos clients and servers.

**seal** To encipher a record containing several fields in such a way that the fields cannot be individually replaced without either 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 is derived from a password.

**server** A particular principal that provides a resource to network clients.

**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.

**subsession 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.

**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.
A
ACME, 62
Administrative utilities, 71
Authentication path, 39
Authentication service, 41

C
Cerberus, 37
Client programs, 67
Compiling Kerberos application, 81
Configuration logs, 47

D
Database, 41
Denial of service attacks, 39

E
Example programs, 82
GSSAPI, 82
KRB5 API, 83

G
GSSAPI example program, 82

I
Installation logs, 50
Inter-realm key, 39

K
kadmin, 41, 71
kdb5_util, 42, 72
KDC, 37, 40
kdestroy, 42, 70
Kerberos
compiling application, 81
linking application, 81
Kerberos ACME, 62
Kerberos database, 41
Kerberos for OpenVMS website, 43
Key Distribution Center, 37
kinit, 41, 67
klist, 42, 69
kpasswd, 42, 71
kprop
using to propagate database, 76
KRB5 API example program, 83
ktutil, 75

L
Linking Kerberos application, 81

M
Massachusetts Institute of Technology, 37
Master KDC server
propagation of, 76

P
Principal name, 37

Private key, 38

R
Realm, 37, 39

S
Secret key, 38
Secret key cryptography, 38
Secure Delivery, 44
Service key, 38
Session key, 38
SSH, 54

T
TCP/IP Services for OpenVMS, 43
Telnet, 58
TGT, 38
Ticket, 38
Ticket-granting service, 41
Ticket-granting ticket, 38

U
Utilities
administrative, 71
user, 67
Utility programs, 41