# GNU Shishi API Reference Manual

| COLLABORATORS |  |  |  |
|---|---|---|---|
|  | *TITLE* : <br><br> GNU Shishi API Reference Manual |  |  |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY |  | August 7, 2022 |  |

| REVISION HISTORY |  |  |  |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
|  |  |  |  |

# Contents

# List of Figures

# Chapter 1

# GNU Shishi API Reference Manual

Shishi is an implementation of the Kerberos 5 network authentication system, as specified in RFC 4120. Shishi can be used to authenticate users in distributed systems.

Shishi contains a library ('libshishi') that can be used by application developers to add support for Kerberos 5. Shishi contains a command line utility ('shishi') that is used by users to acquire and manage tickets (and more). The server side, a Key Distribution Center, is implemented by 'shishid'. Of course, a manual documenting usage aspects as well as the programming API is included.

Shishi currently supports AS/TGS exchanges for acquiring tickets, pre-authentication, the AP exchange for performing client and server authentication, and SAFE/PRIV for integrity/privacy protected application data exchanges.

Shishi is internationalized; error and status messages can be translated into the users' language; user name and passwords can be converted into any available character set (normally including ISO-8859-1 and UTF-8) and also be processed using an experimental Stringprep profile.

Most, if not all, of the widely used encryption and checksum types are supported, such as 3DES, AES, ARCFOUR and HMAC-SHA1.

Shishi is developed for the GNU/Linux system, but runs on over 20 platforms including most major Unix platforms and Windows, and many kind of devices including iPAQ handhelds and S/390 mainframes.

Shishi is free software licensed under the GNU General Public License version 3.0 (or later).

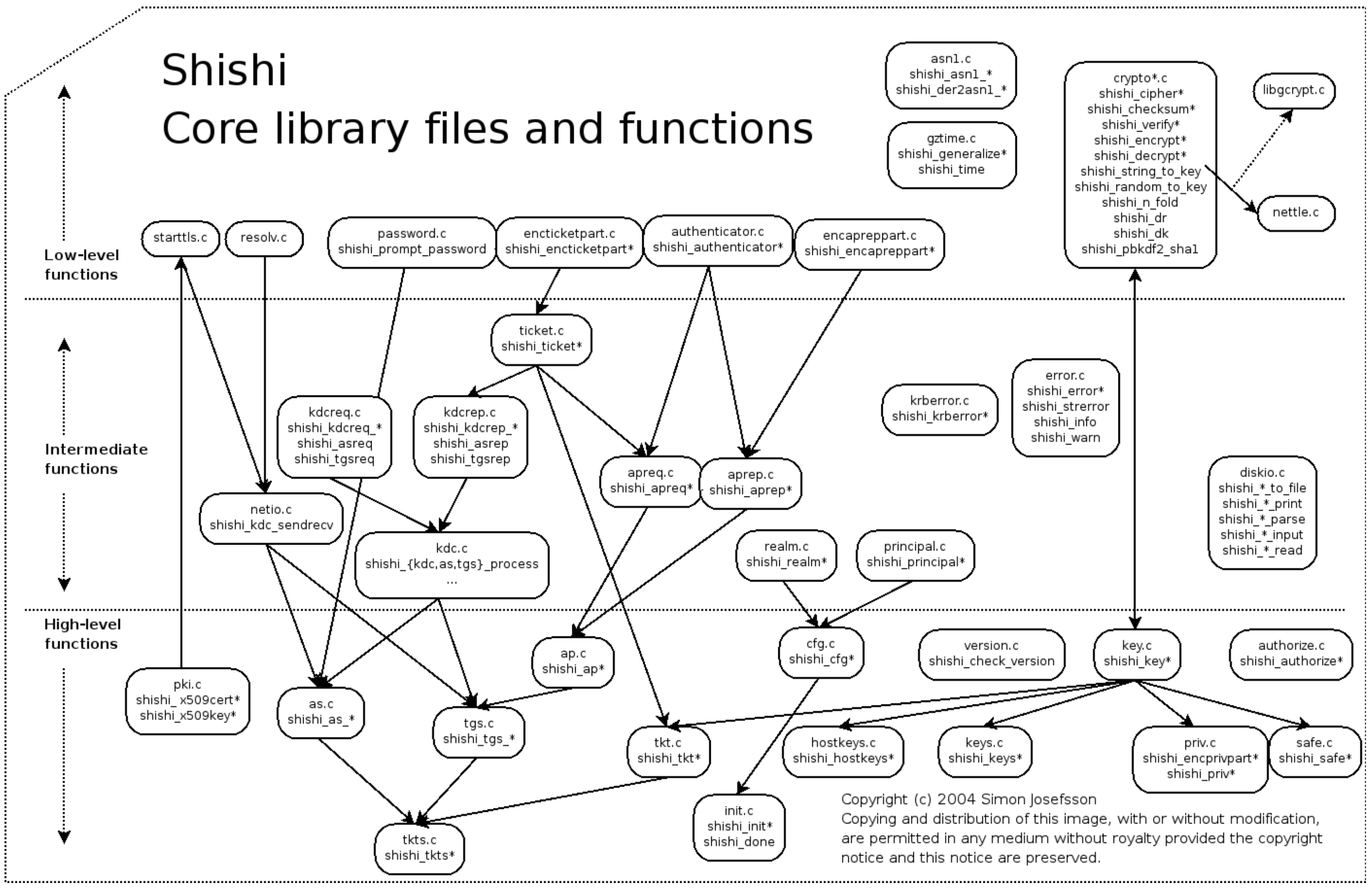
Shishi
Core library files and functions
Low-level functions
Intermediate functions
High-level functions
asn1.c shishi_asn1_* shishi_der2asn1_*
gztime.c shishi_generalize* shishi_time
crypto*.c shishi_cipher* shishi_checksum* shishi_verify* shishi_encrypt* shishi_decrypt* shishi_string_to_key shishi_random_to_key shishi_n_fold shishi_dr shishi_dk shishi_pbkdf2_sha1
libgcrypt.c
nettle.c
starttls.c
resolv.c
password.c shishi_prompt_password
encticketpart.c shishi_encticketpart*
authenticator.c shishi_authenticator*
encapreppart.c shishi_encapreppart*
ticket.c shishi_ticket*
kdcreq.c shishi_kdcreq_* shishi_asreq shishi_tgsreq
kdcrep.c shishi_kdcrep_* shishi_asrep shishi_tgsrep
krberror.c shishi_krberror*
error.c shishi_error* shishi_strerror shishi_info shishi_warn
apreq.c shishi_apreq*
aprep.c shishi_aprep*
diskio.c shishi_*_to_file shishi_*_print shishi_*_parse shishi_*_input shishi_*_read
netio.c shishi_kdc_sendrecv
kdc.c shishi_{kdc,as,tgs}_process ...
realm.c shishi_realm*
principal.c shishi_principal*
ap.c shishi_ap*
cfg.c shishi_cfg*
version.c shishi_check_version
key.c shishi_key*
authorize.c shishi_authorize*
pki.c shishi_x509cert* shishi_x509key*
as.c shishi_as_*
tgs.c shishi_tgs_*
tkt.c shishi_tkt*
hostkeys.c shishi_hostkeys*
keys.c shishi_keys*
priv.c shishi_encprivpart* shishi_priv*
safe.c shishi_safe*
init.c shishi_init* shishi_done
tkts.c shishi_tkts*
Copyright (c) 2004 Simon Josefsson
Copying and distribution of this image, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.

Figure 1.1: Source code layout of Shishi

## 1.1 shishi.h

shishi.h — main library interfaces

### Functions

| | |
|---|---|
| void | (*shishi_alloc_fail_function) () |
| Shishi * | shishi () |
| Shishi * | shishi_server () |
| void | shishi_done () |
| int | shishi_init () |
| int | shishi_init_with_paths () |
| int | shishi_init_server () |
| int | shishi_init_server_with_paths () |
| int | shishi_cfg () |
| int | shishi_cfg_from_file () |
| int | shishi_cfg_print () |
| const char * | shishi_cfg_default_systemfile () |
| const char * | shishi_cfg_default_userdirectory () |
| const char * | shishi_cfg_default_userfile () |
| char * | shishi_cfg_userdirectory_file () |
| int | shishi_cfg_clientkdcetype () |

| | |
|---|---|
| int32_t | shishi_cfg_clientkdcetype_fast () |
| int | shishi_cfg_clientkdcetype_set () |
| int | shishi_cfg_authorizationtype_set () |
| const char * | shishi_strerror () |
| const char * | shishi_error () |
| void | shishi_error_clear () |
| void | shishi_error_set () |
| void | shishi_error_printf () |
| int | shishi_error_outputtype () |
| void | shishi_error_set_outputtype () |
| void | shishi_info () |
| void | shishi_warn () |
| void | shishi_verbose () |
| char * | shishi_realm_default_guess () |
| const char * | shishi_realm_default () |
| void | shishi_realm_default_set () |
| char * | shishi_realm_for_server_file () |
| char * | shishi_realm_for_server_dns () |
| char * | shishi_realm_for_server () |
| char * | shishi_principal_default_guess () |
| const char * | shishi_principal_default () |
| void | shishi_principal_default_set () |
| int | shishi_principal_name () |
| int | shishi_principal_name_realm () |
| int | shishi_principal_name_set () |
| int | shishi_principal_set () |
| int | shishi_parse_name () |
| int | shishi_derive_default_salt () |
| char * | shishi_server_for_local_service () |
| Shishi_asn1 | shishi_ticket () |
| int | shishi_ticket_server () |
| int | shishi_ticket_sname_set () |
| int | shishi_ticket_srealmserver_set () |
| int | shishi_ticket_set_server () |
| int | shishi_ticket_realm_get () |
| int | shishi_ticket_realm_set () |
| int | shishi_ticket_get_enc_part_etype () |
| int | shishi_ticket_set_enc_part () |
| int | shishi_ticket_add_enc_part () |
| int | shishi_ticket_decrypt () |
| Shishi_asn1 | shishi_tkt_ticket () |
| void | shishi_tkt_ticket_set () |
| Shishi_asn1 | shishi_tkt_kdcrep () |
| Shishi_asn1 | shishi_tkt_enckdcreppart () |
| void | shishi_tkt_enckdcreppart_set () |
| Shishi_asn1 | shishi_tkt_encticketpart () |
| void | shishi_tkt_encticketpart_set () |
| Shishi_key * | shishi_tkt_key () |
| int | shishi_tkt_key_set () |
| int | shishi_tkt () |
| Shishi_tkt * | shishi_tkt2 () |
| void | shishi_tkt_pretty_print () |
| int | shishi_tkt_realm () |
| int | shishi_tkt_client () |
| int | shishi_tkt_client_p () |
| int | shishi_tkt_clientrealm () |
| int | shishi_tkt_clientrealm_p () |

| | |
|---|---|
| int | shishi_tkt_clientrealm_set () |
| int | shishi_tkt_serverrealm_set () |
| int | shishi_tkt_build () |
| int | shishi_tkt_lastreq () |
| time_t | shishi_tkt_lastreqc () |
| void | shishi_tkt_lastreq_pretty_print () |
| int | shishi_tkt_authtime () |
| time_t | shishi_tkt_authctime () |
| int | shishi_tkt_starttime () |
| time_t | shishi_tkt_startctime () |
| int | shishi_tkt_endtime () |
| time_t | shishi_tkt_endctime () |
| int | shishi_tkt_renew_till () |
| time_t | shishi_tkt_renew_tillc () |
| int | shishi_tkt_keytype () |
| int32_t | shishi_tkt_keytype_fast () |
| int | shishi_tkt_keytype_p () |
| int | shishi_tkt_server () |
| int | shishi_tkt_server_p () |
| int | shishi_tkt_valid_at_time_p () |
| int | shishi_tkt_valid_now_p () |
| int | shishi_tkt_expired_p () |
| int | shishi_tkt_decrypt () |
| void | shishi_tkt_done () |
| int | shishi_tkt_flags () |
| int | shishi_tkt_flags_set () |
| int | shishi_tkt_flags_add () |
| int | shishi_tkt_forwardable_p () |
| int | shishi_tkt_forwarded_p () |
| int | shishi_tkt_proxiable_p () |
| int | shishi_tkt_proxy_p () |
| int | shishi_tkt_may_postdate_p () |
| int | shishi_tkt_postdated_p () |
| int | shishi_tkt_invalid_p () |
| int | shishi_tkt_renewable_p () |
| int | shishi_tkt_initial_p () |
| int | shishi_tkt_pre_authent_p () |
| int | shishi_tkt_hw_authent_p () |
| int | shishi_tkt_transited_policy_checked_p () |
| int | shishi_tkt_ok_as_delegate_p () |
| char * | shishi_tkts_default_file_guess () |
| const char * | shishi_tkts_default_file () |
| void | shishi_tkts_default_file_set () |
| Shishi_tkts * | shishi_tkts_default () |
| int | shishi_tkts_default_to_file () |
| int | shishi_tkts () |
| Shishi_tkt * | shishi_tkts_nth () |
| int | shishi_tkts_size () |
| int | shishi_tkts_add () |
| int | shishi_tkts_new () |
| int | shishi_tkts_remove () |
| int | shishi_tkts_expire () |
| int | shishi_tkts_print_for_service () |
| int | shishi_tkts_print () |
| int | shishi_tkts_write () |
| int | shishi_tkts_to_file () |

| | |
|---|---|
| int | shishi_tkts_read () |
| int | shishi_tkts_from_file () |
| void | shishi_tkts_done () |
| int | shishi_tkt_match_p () |
| Shishi_tkt * | shishi_tkts_find () |
| Shishi_tkt * | shishi_tkts_find_for_clientserver () |
| Shishi_tkt * | shishi_tkts_find_for_server () |
| Shishi_tkt * | shishi_tkts_get () |
| Shishi_tkt * | shishi_tkts_get_tgt () |
| Shishi_tkt * | shishi_tkts_get_tgs () |
| Shishi_tkt * | shishi_tkts_get_for_clientserver () |
| Shishi_tkt * | shishi_tkts_get_for_server () |
| Shishi_tkt * | shishi_tkts_get_for_localservicepasswd () |
| char * | shishi_tkts_default_ccache_guess () |
| const char * | shishi_tkts_default_ccache () |
| void | shishi_tkts_default_ccache_set () |
| int | shishi_tkts_add_ccache_mem () |
| int | shishi_tkts_add_ccache_file () |
| int | shishi_tkts_from_ccache_mem () |
| int | shishi_tkts_from_ccache_file () |
| int | shishi_enckdcreppart_print () |
| int | shishi_enckdcreppart_save () |
| int | shishi_enckdcreppart_parse () |
| int | shishi_enckdcreppart_read () |
| int | shishi_ticket_save () |
| int | shishi_ticket_print () |
| int | shishi_kdc_print () |
| int | shishi_ticket_parse () |
| int | shishi_ticket_read () |
| int | shishi_etype_info_print () |
| int | shishi_etype_info2_print () |
| int | shishi_padata_print () |
| int | shishi_methoddata_print () |
| Shishi_asn1 | shishi_authenticator () |
| int | shishi_authenticator_set_crealm () |
| int | shishi_authenticator_set_cname () |
| int | shishi_authenticator_client_set () |
| int | shishi_authenticator_ctime () |
| int | shishi_authenticator_ctime_set () |
| int | shishi_authenticator_cusec_get () |
| int | shishi_authenticator_cusec_set () |
| int | shishi_authenticator_seqnumber_get () |
| int | shishi_authenticator_seqnumber_remove () |
| int | shishi_authenticator_seqnumber_set () |
| int | shishi_authenticator_client () |
| int | shishi_authenticator_clientrealm () |
| int | shishi_authenticator_remove_cksum () |
| int | shishi_authenticator_cksum () |
| int | shishi_authenticator_set_cksum () |
| int | shishi_authenticator_add_cksum () |
| int | shishi_authenticator_add_cksum_type () |
| int | shishi_authenticator_remove_subkey () |
| Shishi_asn1 | shishi_authenticator_subkey () |
| int | shishi_authenticator_get_subkey () |
| int | shishi_authenticator_set_subkey () |
| int | shishi_authenticator_add_random_subkey () |
| int | shishi_authenticator_add_random_subkey_etype () |

| | |
|---|---|
| int | shishi_authenticator_add_subkey () |
| int | shishi_authenticator_clear_authorizationdata () |
| int | shishi_authenticator_add_authorizationdata () |
| int | shishi_authenticator_authorizationdata () |
| int | shishi_authenticator_read () |
| int | shishi_authenticator_parse () |
| int | shishi_authenticator_from_file () |
| int | shishi_authenticator_print () |
| int | shishi_authenticator_to_file () |
| int | shishi_authenticator_save () |
| int | shishi_as () |
| void | shishi_as_done () |
| Shishi_asn1 | shishi_as_req () |
| int | shishi_as_req_build () |
| void | shishi_as_req_set () |
| int | shishi_as_req_der () |
| int | shishi_as_req_der_set () |
| Shishi_asn1 | shishi_as_rep () |
| void | shishi_as_rep_set () |
| int | shishi_as_rep_build () |
| int | shishi_as_rep_der () |
| int | shishi_as_rep_der_set () |
| Shishi_asn1 | shishi_as_krberror () |
| int | shishi_as_krberror_der () |
| void | shishi_as_krberror_set () |
| Shishi_tkt * | shishi_as_tkt () |
| void | shishi_as_tkt_set () |
| int | shishi_as_sendrecv () |
| int | shishi_as_sendrecv_hint () |
| int | shishi_as_rep_process () |
| int | shishi_tgs () |
| void | shishi_tgs_done () |
| Shishi_tkt * | shishi_tgs_tgtkt () |
| void | shishi_tgs_tgtkt_set () |
| Shishi_ap * | shishi_tgs_ap () |
| Shishi_asn1 | shishi_tgs_req () |
| int | shishi_tgs_req_der () |
| int | shishi_tgs_req_der_set () |
| void | shishi_tgs_req_set () |
| int | shishi_tgs_req_build () |
| int | shishi_tgs_req_process () |
| Shishi_asn1 | shishi_tgs_rep () |
| int | shishi_tgs_rep_der () |
| int | shishi_tgs_rep_build () |
| int | shishi_tgs_rep_process () |
| Shishi_asn1 | shishi_tgs_krberror () |
| int | shishi_tgs_krberror_der () |
| void | shishi_tgs_krberror_set () |
| Shishi_tkt * | shishi_tgs_tkt () |
| void | shishi_tgs_tkt_set () |
| int | shishi_tgs_sendrecv () |
| int | shishi_tgs_sendrecv_hint () |
| int | shishi_tgs_set_server () |
| int | shishi_tgs_set_realm () |
| int | shishi_tgs_set_realmserver () |
| int | shishi_kdcreq () |

| | |
|---|---|
| Shishi_asn1 | shishi_asreq () |
| Shishi_asn1 | shishi_asreq_rsc () |
| Shishi_asn1 | shishi_tgsreq () |
| Shishi_asn1 | shishi_tgsreq_rst () |
| int | shishi_kdcreq_save () |
| int | shishi_kdcreq_print () |
| int | shishi_kdcreq_to_file () |
| int | shishi_kdcreq_parse () |
| int | shishi_kdcreq_read () |
| int | shishi_kdcreq_from_file () |
| int | shishi_asreq_clientrealm () |
| int | shishi_kdcreq_nonce () |
| int | shishi_kdcreq_nonce_set () |
| int | shishi_kdcreq_client () |
| int | shishi_kdcreq_set_cname () |
| int | shishi_kdcreq_server () |
| int | shishi_kdcreq_set_sname () |
| int | shishi_kdcreq_realm () |
| int | shishi_kdcreq_realm_get () |
| int | shishi_kdcreq_set_realm () |
| int | shishi_kdcreq_set_server () |
| int | shishi_kdcreq_set_realmserver () |
| int | shishi_kdcreq_till () |
| time_t | shishi_kdcreq_tillc () |
| int | shishi_kdcreq_etype () |
| int | shishi_kdcreq_set_etype () |
| int | shishi_kdcreq_options () |
| int | shishi_kdcreq_forwardable_p () |
| int | shishi_kdcreq_forwarded_p () |
| int | shishi_kdcreq_proxiable_p () |
| int | shishi_kdcreq_proxy_p () |
| int | shishi_kdcreq_allow_postdate_p () |
| int | shishi_kdcreq_postdated_p () |
| int | shishi_kdcreq_renewable_p () |
| int | shishi_kdcreq_disable_transited_check_p () |
| int | shishi_kdcreq_renewable_ok_p () |
| int | shishi_kdcreq_enc_tkt_in_skey_p () |
| int | shishi_kdcreq_renew_p () |
| int | shishi_kdcreq_validate_p () |
| int | shishi_kdcreq_options_set () |
| int | shishi_kdcreq_options_add () |
| int | shishi_kdcreq_clear_padata () |
| int | shishi_kdcreq_get_padata () |
| int | shishi_kdcreq_get_padata_tgs () |
| int | shishi_kdcreq_add_padata () |
| int | shishi_kdcreq_add_padata_tgs () |
| int | shishi_kdcreq_add_padata_preauth () |
| int | shishi_kdcreq_build () |
| int | shishi_as_derive_salt () |
| int | shishi_tgs_process () |
| int | shishi_as_process () |
| int | shishi_kdc_process () |
| int | shishi_kdcreq_sendrecv () |
| int | shishi_kdcreq_sendrecv_hint () |
| int | shishi_kdc_copy_crealm () |
| int | shishi_as_check_crealm () |
| int | shishi_kdc_copy_cname () |

| | |
|---|---|
| int | shishi_as_check_cname () |
| int | shishi_kdc_copy_nonce () |
| int | shishi_kdc_check_nonce () |
| Shishi_asn1 | shishi_asrep () |
| Shishi_asn1 | shishi_tgsrep () |
| int | shishi_kdcrep_save () |
| int | shishi_kdcrep_print () |
| int | shishi_kdcrep_to_file () |
| int | shishi_kdcrep_parse () |
| int | shishi_kdcrep_read () |
| int | shishi_kdcrep_from_file () |
| int | shishi_kdcrep_clear_padata () |
| int | shishi_kdcrep_get_enc_part_etype () |
| int | shishi_kdcrep_add_enc_part () |
| int | shishi_kdcrep_get_ticket () |
| int | shishi_kdcrep_set_ticket () |
| int | shishi_kdcrep_crealm_set () |
| int | shishi_kdcrep_cname_set () |
| int | shishi_kdcrep_client_set () |
| int | shishi_kdcrep_crealmserver_set () |
| int | shishi_kdcrep_set_enc_part () |
| int | shishi_kdcrep_decrypt () |
| Shishi_asn1 | shishi_enckdcreppart () |
| Shishi_asn1 | shishi_encasreppart () |
| int | shishi_enckdcreppart_get_key () |
| int | shishi_enckdcreppart_key_set () |
| int | shishi_enckdcreppart_nonce_set () |
| int | shishi_enckdcreppart_flags_set () |
| int | shishi_enckdcreppart_authtime_set () |
| int | shishi_enckdcreppart_starttime_set () |
| int | shishi_enckdcreppart_endtime_set () |
| int | shishi_enckdcreppart_renew_till_set () |
| int | shishi_enckdcreppart_srealm_set () |
| int | shishi_enckdcreppart_sname_set () |
| int | shishi_enckdcreppart_server_set () |
| int | shishi_enckdcreppart_srealmserver_set () |
| int | shishi_enckdcreppart_populate_encticketpart () |
| Shishi_asn1 | shishi_krberror () |
| int | shishi_krberror_print () |
| int | shishi_krberror_save () |
| int | shishi_krberror_to_file () |
| int | shishi_krberror_parse () |
| int | shishi_krberror_read () |
| int | shishi_krberror_from_file () |
| int | shishi_krberror_build () |
| int | shishi_krberror_der () |
| int | shishi_krberror_crealm () |
| int | shishi_krberror_remove_crealm () |
| int | shishi_krberror_set_crealm () |
| int | shishi_krberror_client () |
| int | shishi_krberror_set_cname () |
| int | shishi_krberror_remove_cname () |
| int | shishi_krberror_client_set () |
| int | shishi_krberror_realm () |
| int | shishi_krberror_set_realm () |
| int | shishi_krberror_server () |

| | |
|---|---|
| int | shishi_krberror_remove_sname () |
| int | shishi_krberror_set_sname () |
| int | shishi_krberror_server_set () |
| int | shishi_krberror_ctime () |
| int | shishi_krberror_ctime_set () |
| int | shishi_krberror_remove_ctime () |
| int | shishi_krberror_cusec () |
| int | shishi_krberror_cusec_set () |
| int | shishi_krberror_remove_cusec () |
| int | shishi_krberror_stime () |
| int | shishi_krberror_stime_set () |
| int | shishi_krberror_susec () |
| int | shishi_krberror_susec_set () |
| int | shishi_krberror_errorcode_set () |
| int | shishi_krberror_etext () |
| int | shishi_krberror_set_etext () |
| int | shishi_krberror_remove_etext () |
| int | shishi_krberror_edata () |
| int | shishi_krberror_set_edata () |
| int | shishi_krberror_remove_edata () |
| int | shishi_krberror_errorcode () |
| int | shishi_krberror_errorcode_fast () |
| int | shishi_krberror_pretty_print () |
| const char * | shishi_krberror_errorcode_message () |
| const char * | shishi_krberror_message () |
| int | shishi_krberror_methoddata () |
| const char * | shishi_generalize_time () |
| const char * | shishi_generalize_now () |
| time_t | shishi_generalize_ctime () |
| int | shishi_time () |
| int | shishi_ctime () |
| int | shishi_randomize () |
| int | shishi_crc () |
| int | shishi_md4 () |
| int | shishi_md5 () |
| int | shishi_hmac_md5 () |
| int | shishi_hmac_sha1 () |
| int | shishi_des_cbc_mac () |
| int | shishi_arcfour () |
| int | shishi_des () |
| int | shishi_3des () |
| int | shishi_aes_cts () |
| int | shishi_cipher_supported_p () |
| const char * | shishi_cipher_name () |
| int | shishi_cipher_blocksize () |
| int | shishi_cipher_confoundersize () |
| size_t | shishi_cipher_keylen () |
| size_t | shishi_cipher_randomlen () |
| int | shishi_cipher_defaultcksumtype () |
| int | shishi_cipher_parse () |
| int | shishi_checksum_supported_p () |
| const char * | shishi_checksum_name () |
| size_t | shishi_checksum_cksumlen () |
| int | shishi_checksum_parse () |
| int | shishi_string_to_key () |
| int | shishi_random_to_key () |
| int | shishi_encrypt_ivupdate_etype () |

| | |
|---|---|
| int | shishi_encrypt_iv_etype () |
| int | shishi_encrypt_etype () |
| int | shishi_encrypt_ivupdate () |
| int | shishi_encrypt_iv () |
| int | shishi_encrypt () |
| int | shishi_decrypt_ivupdate_etype () |
| int | shishi_decrypt_iv_etype () |
| int | shishi_decrypt_etype () |
| int | shishi_decrypt_ivupdate () |
| int | shishi_decrypt_iv () |
| int | shishi_decrypt () |
| int | shishi_checksum () |
| int | shishi_verify () |
| int | shishi_dk () |
| int | shishi_dr () |
| int | shishi_n_fold () |
| int | shishi_pbkdf2_sha1 () |
| Shishi_crypto * | shishi_crypto () |
| void | shishi_crypto_close () |
| int | shishi_crypto_encrypt () |
| int | shishi_crypto_decrypt () |
| const char * | shishi_check_version () |
| int | (*shishi_prompt_password_func) () |
| void | shishi_prompt_password_callback_set () |
| shishi_prompt_password_func | shishi_prompt_password_callback_get () |
| int | shishi_prompt_password () |
| int | shishi_asn1_number_of_elements () |
| int | shishi_asn1_empty_p () |
| int | shishi_asn1_read () |
| int | shishi_asn1_read_inline () |
| int | shishi_asn1_read_integer () |
| int | shishi_asn1_read_int32 () |
| int | shishi_asn1_read_uint32 () |
| int | shishi_asn1_read_bitstring () |
| int | shishi_asn1_read_optional () |
| int | shishi_asn1_write () |
| int | shishi_asn1_write_integer () |
| int | shishi_asn1_write_int32 () |
| int | shishi_asn1_write_uint32 () |
| int | shishi_asn1_write_bitstring () |
| void | shishi_asn1_done () |
| Shishi_asn1 | shishi_asn1_pa_enc_ts_enc () |
| Shishi_asn1 | shishi_asn1_encrypteddata () |
| Shishi_asn1 | shishi_asn1_padata () |
| Shishi_asn1 | shishi_asn1_methoddata () |
| Shishi_asn1 | shishi_asn1_etype_info () |
| Shishi_asn1 | shishi_asn1_etype_info2 () |
| Shishi_asn1 | shishi_asn1_asreq () |
| Shishi_asn1 | shishi_asn1_asrep () |
| Shishi_asn1 | shishi_asn1_tgsreq () |
| Shishi_asn1 | shishi_asn1_tgsrep () |
| Shishi_asn1 | shishi_asn1_apreq () |
| Shishi_asn1 | shishi_asn1_aprep () |
| Shishi_asn1 | shishi_asn1_ticket () |
| Shishi_asn1 | shishi_asn1_encapreppart () |
| Shishi_asn1 | shishi_asn1_encticketpart () |

| | |
|---|---|
| Shishi_asn1 | shishi_asn1_authenticator () |
| Shishi_asn1 | shishi_asn1_enckdcreppart () |
| Shishi_asn1 | shishi_asn1_encasreppart () |
| Shishi_asn1 | shishi_asn1_krberror () |
| Shishi_asn1 | shishi_asn1_krbsafe () |
| Shishi_asn1 | shishi_asn1_priv () |
| Shishi_asn1 | shishi_asn1_encprivpart () |
| int | shishi_asn1_to_der () |
| int | shishi_asn1_to_der_field () |
| Shishi_msgtype | shishi_asn1_msgtype () |
| Shishi_msgtype | shishi_der_msgtype () |
| void | shishi_asn1_print () |
| Shishi_asn1 | shishi_der2asn1 () |
| Shishi_asn1 | shishi_der2asn1_padata () |
| Shishi_asn1 | shishi_der2asn1_methoddata () |
| Shishi_asn1 | shishi_der2asn1_etype_info () |
| Shishi_asn1 | shishi_der2asn1_etype_info2 () |
| Shishi_asn1 | shishi_der2asn1_ticket () |
| Shishi_asn1 | shishi_der2asn1_encticketpart () |
| Shishi_asn1 | shishi_der2asn1_asreq () |
| Shishi_asn1 | shishi_der2asn1_tgsreq () |
| Shishi_asn1 | shishi_der2asn1_asrep () |
| Shishi_asn1 | shishi_der2asn1_tgsrep () |
| Shishi_asn1 | shishi_der2asn1_kdcrep () |
| Shishi_asn1 | shishi_der2asn1_kdcreq () |
| Shishi_asn1 | shishi_der2asn1_apreq () |
| Shishi_asn1 | shishi_der2asn1_aprep () |
| Shishi_asn1 | shishi_der2asn1_authenticator () |
| Shishi_asn1 | shishi_der2asn1_krberror () |
| Shishi_asn1 | shishi_der2asn1_krbsafe () |
| Shishi_asn1 | shishi_der2asn1_priv () |
| Shishi_asn1 | shishi_der2asn1_encasreppart () |
| Shishi_asn1 | shishi_der2asn1_enctgsreppart () |
| Shishi_asn1 | shishi_der2asn1_enckdcreppart () |
| Shishi_asn1 | shishi_der2asn1_encapreppart () |
| Shishi_asn1 | shishi_der2asn1_encprivpart () |
| int | shishi_ap () |
| int | shishi_ap_etype () |
| int | shishi_ap_nosubkey () |
| void | shishi_ap_done () |
| int | shishi_ap_set_tktoptions () |
| int | shishi_ap_tktoptions () |
| int | shishi_ap_etype_tktoptionsdata () |
| int | shishi_ap_set_tktoptionsdata () |
| int | shishi_ap_tktoptionsdata () |
| int | shishi_ap_set_tktoptionsraw () |
| int | shishi_ap_tktoptionsraw () |
| int | shishi_ap_set_tktoptionsasn1usage () |
| int | shishi_ap_tktoptionsasn1usage () |
| Shishi_tkt * | shishi_ap_tkt () |
| void | shishi_ap_tkt_set () |
| int | shishi_ap_authenticator_cksumdata () |
| void | shishi_ap_authenticator_cksumdata_set () |
| void | shishi_ap_authenticator_cksumraw_set () |
| int32_t | shishi_ap_authenticator_cksumtype () |
| void | shishi_ap_authenticator_cksumtype_set () |
| Shishi_asn1 | shishi_ap_authenticator () |

| | |
|---|---|
| void | shishi_ap_authenticator_set () |
| Shishi_asn1 | shishi_ap_req () |
| void | shishi_ap_req_set () |
| int | shishi_ap_req_der () |
| int | shishi_ap_req_der_set () |
| int | shishi_ap_req_build () |
| int | shishi_ap_req_asn1 () |
| Shishi_key * | shishi_ap_key () |
| int | shishi_ap_req_decode () |
| int | shishi_ap_req_process () |
| int | shishi_ap_req_process_keyusage () |
| Shishi_asn1 | shishi_ap_rep () |
| void | shishi_ap_rep_set () |
| int | shishi_ap_rep_der () |
| int | shishi_ap_rep_der_set () |
| int | shishi_ap_rep_verify () |
| int | shishi_ap_rep_verify_der () |
| int | shishi_ap_rep_verify_asn1 () |
| int | shishi_ap_rep_asn1 () |
| int | shishi_ap_rep_build () |
| Shishi_asn1 | shishi_ap_encapreppart () |
| void | shishi_ap_encapreppart_set () |
| const char * | shishi_ap_option2string () |
| Shishi_apoptions | shishi_ap_string2option () |
| const char * | shishi_key_principal () |
| void | shishi_key_principal_set () |
| const char * | shishi_key_realm () |
| void | shishi_key_realm_set () |
| int | shishi_key_type () |
| void | shishi_key_type_set () |
| const char * | shishi_key_value () |
| void | shishi_key_value_set () |
| const char * | shishi_key_name () |
| size_t | shishi_key_length () |
| uint32_t | shishi_key_version () |
| void | shishi_key_version_set () |
| time_t | shishi_key_timestamp () |
| void | shishi_key_timestamp_set () |
| int | shishi_key () |
| void | shishi_key_done () |
| void | shishi_key_copy () |
| int | shishi_key_print () |
| int | shishi_key_to_file () |
| int | shishi_key_parse () |
| int | shishi_key_random () |
| int | shishi_key_from_value () |
| int | shishi_key_from_base64 () |
| int | shishi_key_from_random () |
| int | shishi_key_from_string () |
| int | shishi_key_from_name () |
| int | shishi_keys () |
| void | shishi_keys_done () |
| int | shishi_keys_size () |
| const Shishi_key * | shishi_keys_nth () |
| void | shishi_keys_remove () |
| int | shishi_keys_add () |

| | |
|---|---|
| int | shishi_keys_print () |
| int | shishi_keys_from_file () |
| int | shishi_keys_to_file () |
| Shishi_key * | shishi_keys_for_serverrealm_in_file () |
| Shishi_key * | shishi_keys_for_server_in_file () |
| Shishi_key * | shishi_keys_for_localservicerealm_in_file () |
| int | shishi_keys_add_keytab_mem () |
| int | shishi_keys_add_keytab_file () |
| int | shishi_keys_from_keytab_mem () |
| int | shishi_keys_from_keytab_file () |
| int | shishi_keys_to_keytab_mem () |
| int | shishi_keys_to_keytab_file () |
| const char * | shishi_hostkeys_default_file () |
| void | shishi_hostkeys_default_file_set () |
| Shishi_key * | shishi_hostkeys_for_server () |
| Shishi_key * | shishi_hostkeys_for_serverrealm () |
| Shishi_key * | shishi_hostkeys_for_localservicerealm () |
| Shishi_key * | shishi_hostkeys_for_localservice () |
| Shishi_asn1 | shishi_encapreppart () |
| int | shishi_encapreppart_time_copy () |
| int | shishi_encapreppart_ctime () |
| int | shishi_encapreppart_ctime_set () |
| int | shishi_encapreppart_cusec_get () |
| int | shishi_encapreppart_cusec_set () |
| int | shishi_encapreppart_print () |
| int | shishi_encapreppart_save () |
| int | shishi_encapreppart_to_file () |
| int | shishi_encapreppart_read () |
| int | shishi_encapreppart_parse () |
| int | shishi_encapreppart_from_file () |
| int | shishi_encapreppart_get_key () |
| int | shishi_encapreppart_seqnumber_get () |
| int | shishi_encapreppart_seqnumber_remove () |
| int | shishi_encapreppart_seqnumber_set () |
| Shishi_asn1 | shishi_apreq () |
| int | shishi_apreq_parse () |
| int | shishi_apreq_from_file () |
| int | shishi_apreq_print () |
| int | shishi_apreq_to_file () |
| int | shishi_apreq_read () |
| int | shishi_apreq_save () |
| int | shishi_apreq_set_ticket () |
| int | shishi_apreq_set_authenticator () |
| int | shishi_apreq_add_authenticator () |
| int | shishi_apreq_options () |
| int | shishi_apreq_use_session_key_p () |
| int | shishi_apreq_mutual_required_p () |
| int | shishi_apreq_options_set () |
| int | shishi_apreq_options_add () |
| int | shishi_apreq_options_remove () |
| int | shishi_apreq_get_ticket () |
| int | shishi_apreq_get_authenticator_etype () |
| int | shishi_apreq_decrypt () |
| Shishi_asn1 | shishi_aprep () |
| int | shishi_aprep_print () |
| int | shishi_aprep_save () |
| int | shishi_aprep_to_file () |

| | |
|---|---|
| int | shishi_aprep_read () |
| int | shishi_aprep_parse () |
| int | shishi_aprep_from_file () |
| int | shishi_aprep_decrypt () |
| int | shishi_aprep_verify () |
| int | shishi_aprep_enc_part_set () |
| int | shishi_aprep_enc_part_add () |
| int | shishi_aprep_enc_part_make () |
| int | shishi_aprep_get_enc_part_etype () |
| int | shishi_kdc_sendrecv () |
| int | shishi_kdc_sendrecv_hint () |
| Shishi_asn1 | shishi_encticketpart () |
| int | shishi_encticketpart_key_set () |
| int | shishi_encticketpart_get_key () |
| int | shishi_encticketpart_crealm () |
| int | shishi_encticketpart_crealm_set () |
| int | shishi_encticketpart_client () |
| int | shishi_encticketpart_clientrealm () |
| int | shishi_encticketpart_cname_set () |
| int | shishi_encticketpart_print () |
| int | shishi_encticketpart_flags_set () |
| int | shishi_encticketpart_transited_set () |
| int | shishi_encticketpart_authtime_set () |
| int | shishi_encticketpart_endtime_set () |
| int | shishi_encticketpart_authtime () |
| time_t | shishi_encticketpart_authctime () |
| int | shishi_safe () |
| void | shishi_safe_done () |
| Shishi_key * | shishi_safe_key () |
| void | shishi_safe_key_set () |
| Shishi_asn1 | shishi_safe_safe () |
| void | shishi_safe_safe_set () |
| int | shishi_safe_safe_der () |
| int | shishi_safe_safe_der_set () |
| int | shishi_safe_print () |
| int | shishi_safe_save () |
| int | shishi_safe_to_file () |
| int | shishi_safe_parse () |
| int | shishi_safe_read () |
| int | shishi_safe_from_file () |
| int | shishi_safe_cksum () |
| int | shishi_safe_set_cksum () |
| int | shishi_safe_user_data () |
| int | shishi_safe_set_user_data () |
| int | shishi_safe_build () |
| int | shishi_safe_verify () |
| int | shishi_priv () |
| void | shishi_priv_done () |
| Shishi_key * | shishi_priv_key () |
| void | shishi_priv_key_set () |
| Shishi_asn1 | shishi_priv_priv () |
| void | shishi_priv_priv_set () |
| int | shishi_priv_priv_der () |
| int | shishi_priv_priv_der_set () |
| Shishi_asn1 | shishi_priv_encprivpart () |
| void | shishi_priv_encprivpart_set () |

| | |
|---|---|
| int | shishi_priv_encprivpart_der () |
| int | shishi_priv_encprivpart_der_set () |
| int | shishi_priv_print () |
| int | shishi_priv_save () |
| int | shishi_priv_to_file () |
| int | shishi_priv_parse () |
| int | shishi_priv_read () |
| int | shishi_priv_from_file () |
| int | shishi_priv_enc_part_etype () |
| int | shishi_priv_set_enc_part () |
| int | shishi_encprivpart_user_data () |
| int | shishi_encprivpart_set_user_data () |
| int | shishi_priv_build () |
| int | shishi_priv_process () |
| int | shishi_authorized_p () |
| int | shishi_authorization_parse () |
| int | shishi_authorize_strcmp () |
| int | shishi_authorize_k5login () |
| char * | shishi_x509ca_default_file_guess () |
| void | shishi_x509ca_default_file_set () |
| const char * | shishi_x509ca_default_file () |
| char * | shishi_x509cert_default_file_guess () |
| void | shishi_x509cert_default_file_set () |
| const char * | shishi_x509cert_default_file () |
| char * | shishi_x509key_default_file_guess () |
| void | shishi_x509key_default_file_set () |
| const char * | shishi_x509key_default_file () |
| time_t | shishi_get_date () |
| void | shishi_xalloc_die () |
| Shishi_dns | shishi_resolv () |
| void | shishi_resolv_free () |

## Types and Values

| | |
|---|---|
| enum | Shishi_rc |
| enum | Shishi_name_type |
| enum | Shishi_padata_type |
| enum | Shishi_tr_type |
| enum | Shishi_apoptions |
| enum | Shishi_ticketflags |
| enum | Shishi_KDCOptions |
| enum | Shishi_msgtype |
| enum | Shishi_lrtype |
| enum | Shishi_etype |
| enum | Shishi_cksumtype |
| enum | Shishi_filetype |
| enum | Shishi_outputtype |
| enum | Shishi_authorization |
| enum | Shishi_keyusage |
| enum | Shishi_krb_error |
| enum | Shishi_tkts_hintflags |
| struct | Shishi_tkts_hint |
| struct | Shishi_dns_st |
| struct | Shishi_dns_srv_st |
| #define | SHISHI_DNS_IN |
| #define | SHISHI_DNS_TXT |

| | |
|---|---|
| #define | SHISHI_DNS_SRV |
| typedef | Shishi_dns |
| typedef | Shishi_dns_srv |
| typedef | Shishi |
| typedef | Shishi_tkt |
| typedef | Shishi_tkts |
| typedef | Shishi_as |
| typedef | Shishi_tgs |
| typedef | Shishi_ap |
| typedef | Shishi_key |
| typedef | Shishi_keys |
| typedef | Shishi_safe |
| typedef | Shishi_priv |
| typedef | Shishi_asn1 |
| typedef | Shishi_crypto |
| #define | SHISHI_GENERALIZEDTIME_LENGTH |
| #define | SHISHI_GENERALIZEDTIMEZ_LENGTH |

## Description

The main library interfaces are declared in shishi.h.

## Functions

### shishi_alloc_fail_function ()

```
void
(*shishi_alloc_fail_function) (void);
```

### shishi ()

```
Shishi~*
shishi (void);
```

Initializes the Shishi library, and primes logging so that future warnings and informational messages are printed on stderr. If this function fails, it may send its own diagnostic errors to stderr.

#### Returns

Returns a Shishi library handle, or NULL on error.

### shishi_server ()

```
Shishi~*
shishi_server (void);
```

Initializes the Shishi library, and primes logging so that future warnings and informational messages are sent to the syslog system. If this function fails, it may print diagnostic errors in the syslog.

#### Returns

Returns a Shishi library handle, or NULL on error.

### shishi_done ()

```
void
shishi_done (Shishi *handle);
```

Deallocates the Shishi library handle. The handle must not be used in any call to a shishi function after an execution of shishi_done().

If there is a default tkts, it is written to the default tkts file. If you do not wish to write the default tkts file, close the default file before calling this function. It is closed with a simple shishi_tkts_done(*handle* , NULL). For related information, see shishi_tkts_default_file_set().

#### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |

### shishi_init ()

```
int
shishi_init (Shishi **handle);
```

Creates a Shishi library handle, using shishi(), and reads the system configuration file, user configuration file and user tickets from their default locations. The paths to the system configuration file is decided at compile time, and is $sysconfdir/shishi.conf. The user configuration file is $HOME/.shishi/config, and the user ticket file is $HOME/.shishi/ticket.

The handle is allocated regardless of return value. The single exception being SHISHI_HANDLE_ERROR, which indicates a problem in allocating the handle. Other error conditions could arise while reading files.

#### Parameters

| | |
|---|---|
| handle | Pointer to a Shishi handle created by this call. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_init_with_paths ()

```
int
shishi_init_with_paths (Shishi **handle,
                        const char *tktsfile,
                        const char *systemcfgfile,
                        const char *usercfgfile);
```

Creates a Shishi library handle, using shishi(), and reads the system configuration file, user configuration file, and user tickets at the specified locations. If any of *usercfgfile* or *systemcfgfile* is NULL, the file is read from its default location, which for the system configuration is decided at compile time, and is $sysconfdir/shishi.conf, and for the user configuration it is $HOME/.shishi/config. If the ticket file name is NULL, a ticket file is not read at all.

The handle is allocated regardless of return value. The single exception being SHISHI_HANDLE_ERROR, which indicates a problem in allocating the handle. Other error conditions could arise while reading files.

#### Parameters

| | | |
|---|---|---|
| handle | Pointer to a Shishi handle created by this call. | |
| tktsfile | Filename of ticket file, or NULL. | |
| systemcfgfile | Filename of system configuration, or NULL. | |
| usercfgfile | Filename of user configuration, or NULL. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_init_server ()

```
int
shishi_init_server (Shishi **handle);
```

Creates a Shishi library handle, using shishi_server(), and reads the system configuration file. The path to the system configuration file is decided at compile time, and is $sysconfdir/shishi.conf.

The handle is allocated regardless of return value. The single exception being SHISHI_HANDLE_ERROR, which indicates a problem in allocating the handle. Other error conditions could arise while reading the file.

#### Parameters

| | | |
|---|---|---|
| handle | Pointer to a Shishi handle created by this call. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_init_server_with_paths ()

```
int
shishi_init_server_with_paths (Shishi **handle,
                               const char *systemcfgfile);
```

Creates a Shishi library handle, using shishi_server(), and reads the system configuration file from the specified location. The path to the system configuration file is decided at compile time, and is $sysconfdir/shishi.conf.

The handle is allocated regardless of return value. The single exception being SHISHI_HANDLE_ERROR, which indicates a problem in allocating the handle. Other error conditions could arise while reading the file.

#### Parameters

| | | |
|---|---|---|
| handle | Pointer to a Shishi handle created by this call. | |
| systemcfgfile | Filename of system configuration, or NULL. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_cfg ()

```
int
shishi_cfg (Shishi *handle,
            const char *option);
```

Configures the shishi library according to the options given in `option` .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| option | String containing shishi library options. | |

#### Returns

Returns SHISHI_OK if `option` is valid and configuration was successful.

### shishi_cfg_from_file ()

```
int
shishi_cfg_from_file (Shishi *handle,
                      const char *cfg);
```

Configures the shishi library using a configuration file located at `cfg` .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| cfg | Name of configuration file. | |

#### Returns

Returns SHISHI_OK if successful.

### shishi_cfg_print ()

```
int
shishi_cfg_print (Shishi *handle,
                  FILE *fh);
```

Prints library configuration status to `fh` . This function is mostly intended for debugging purposes.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| fh | File stream handle opened for writing. |

#### Returns

Always returns SHISHI_OK.

### shishi_cfg_default_systemfile ()

```
const char~*
shishi_cfg_default_systemfile (Shishi *handle);
```

The system configuration file name is decided at compile time, but is replaced by assigning another file name to the environment variable $SHISHI_CONFIG. This call offers a single interface for determining the file name, to which the library turns for its settings.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |

#### Returns

Returns file name of present system configuration.

### shishi_cfg_default_userdirectory ()

```
const char~*
shishi_cfg_default_userdirectory (Shishi *handle);
```

The default user directory, referred to for Shishi ticket cache and other purposes, is normally computed by appending the fixed string "/.shishi" to the content of the environment variable $HOME.

This hard coded directory, i.e., "$HOME/.shishi/", can be replaced by whatever complete path is stored in the environment variable $SHISHI_HOME.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |

#### Returns

Returns the user's directory name where the Shishi library will search for configuration files, ticket caches, etcetera.

### shishi_cfg_default_userfile ()

```
const char~*
shishi_cfg_default_userfile (Shishi *handle);
```

Reports the absolute filename of the default user configuration file. This is typically "$HOME/.shishi/shishi.conf".

The value of $SHISHI_HOME will change the directory part, as stated regarding shishi_cfg_default_userdirectory().

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |

#### Returns

Returns the user's configuration filename.

### shishi_cfg_userdirectory_file ()

```
char~*
shishi_cfg_userdirectory_file (Shishi *handle,
                               const char *file);
```

Reports the full path to the file where the Shishi library expects to find the user's library configuration, given that the file itself is named by the parameter `file` .

The answer is composed from the value of `file` and the directory returned by shishi_cfg_default_userdirectory(). Typically, the returned string would be expanded from "$HOME/.shishi/`file` ".

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| file | Basename of file to use for the user's configuration settings of the library. |

#### Returns

Returns the absolute filename to the argument `file` , relative to the user specific Shishi configuration directory.

### shishi_cfg_clientkdcetype ()

```
int
shishi_cfg_clientkdcetype (Shishi *handle,
                           int32_t **etypes);
```

Sets the variable `etypes` to a static array of preferred encryption types applicable to clients.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| etypes | Pointer to an array of encryption types. |

#### Returns

Returns the number of encryption types referred to by the updated array pointer, or zero, should no type exist.

### shishi_cfg_clientkdcetype_fast ()

```
int32_t
shishi_cfg_clientkdcetype_fast (Shishi *handle);
```

Extracts the default encryption type from the list of preferred encryption types acceptable to the client.

When the preferred list is empty, SHISHI_AES256_CTS_HMAC_SHA1_96 is returned as a sensible default type.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |

#### Returns

Returns the default encryption type.

### shishi_cfg_clientkdcetype_set ()

```
int
shishi_cfg_clientkdcetype_set (Shishi *handle,
                               char *value);
```

Sets the configuration option "client-kdc-etypes" from `value` . The string contains encryption types, integers or names, separated by comma or by whitespace. An example naming three encryption types could be:

aes256-cts-hmac-sha1-96 des3-cbc-sha1-kd des-cbc-md5

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| value | String naming acceptable encryption types. |

#### Returns

Returns SHISHI_OK if successful, and SHISHI_INVALID_ARGUMENT otherwise.

### shishi_cfg_authorizationtype_set ()

```
int
shishi_cfg_authorizationtype_set (Shishi *handle,
                                  char *value);
```

Sets the configuration option "authorization-types" from `value` . The string contains authorization types, integers or names, separated by comma or whitespace.

As an example, "k5login basic" would first check Kerberos5 authentication based on preset principals, and then fall back to the basic test of identical principal names.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| value | String listing acceptable authorization types. |

#### Returns

Returns SHISHI_OK if successful, and SHISHI_INVALID_ARGUMENT otherwise.

### shishi_strerror ()

```
const char~*
shishi_strerror (int err);
```

Converts the return code in *err* to a human readable string.

#### Parameters

| | |
|---|---|
| err | shishi error code. |

#### Returns

Returns a pointer to a statically allocated string containing a description of the error with code *err* . This string can be used to output a diagnostic message to the user.

### shishi_error ()

```
const char~*
shishi_error (Shishi *handle);
```

Extracts detailed information on the most recently occurred error condition. Note that memory is managed by the Shishi library, so the returned string must not be deallocated.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns a pointer to a string describing an error. The string must not be deallocated by the caller.

### shishi_error_clear ()

```
void
shishi_error_clear (Shishi *handle);
```

Clears the internal error description. See shishi_error() on how to access the error string, and shishi_error_set() as well as shishi_error_printf() on how to set the error string.

This function is mostly for Shishi's internal use, but if you develop an extension of Shishi, it may be useful to support the same error handling infrastructure.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

### shishi_error_set ()

```
void
shishi_error_set (Shishi *handle,
                  const char *errstr);
```

Sets the error description to the content of `errstr` . The string is copied into the Shishi internal structure, so you can deallocate any string passed to this function.

This function is mostly for Shishi's internal use, but if you develop an extension of Shishi, it may be useful to support the same error handling infrastructure.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| errstr | A null-terminated character string holding a description, or NULL to clear the internal error string. |

### shishi_error_printf ()

```
void
shishi_error_printf (Shishi *handle,
                     const char *format,
                     ...);
```

Sets the internal error description to a printf(3) formatted string. This function is mostly for Shishi's internal use, but if you develop an extension of Shishi, it may be useful to support the same infrastructure for error handling.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| format | printf style format string. |
| ... | printf style arguments. |

### shishi_error_outputtype ()

```
int
shishi_error_outputtype (Shishi *handle);
```

Reports the current output type used in message logging.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the output type. SHISHI_OUTPUTTYPE_NULL means no output. SHISHI_OUTPUTTYPE_STDERR and SHISHI_OUTPUT
direct text to the console, or to the syslog system.

### shishi_error_set_outputtype ()

```
void
shishi_error_set_outputtype (Shishi *handle,
                             int type);
```

Sets the output type (NULL, stderr or syslog) used for information and warning messages. Intended values are SHISHI_OUTPUTTYPE_
for no output at all, SHISHI_OUTPUTTYPE_STDERR for output to the console, and SHISHI_OUTPUTTYPE_SYSLOG for syslog messaging. The first value covers everything different from the latter two values.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| type | output type, of enum type Shishi_outputtype. |

### shishi_info ()

```
void
shishi_info (Shishi *handle,
             const char *format,
             ...);
```

Prints an informational message, composed from the arguments, to the output stream set in `handle` .

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| format | printf style format string. |
| ... | printf style arguments. |

### shishi_warn ()

```
void
shishi_warn (Shishi *handle,
             const char *format,
             ...);
```

Prints a warning, composed from the arguments, to the output stream set in `handle` .

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| format | printf style format string. | |
| ... | printf style arguments. | |

### shishi_verbose ()

```
void
shishi_verbose (Shishi *handle,
                const char *format,
                ...);
```

Prints a diagnostic message, composed from the arguments, to the output stream set in *handle* . The current verbosity setting determines whether the message is actually printed, or is suppressed due to low significance.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| format | printf style format string. | |
| ... | printf style arguments. | |

### shishi_realm_default_guess ()

```
char~*
shishi_realm_default_guess (void);
```

Guesses a realm based on getdomainname(), which really responds with a NIS/YP domain, but if set properly, it might be a good first guess. If this NIS query fails, call gethostname(), and on its failure, fall back to returning the artificial string "could-not-guess-default-realm".

Note that the hostname is not trimmed off of the string returned by gethostname(), thus pretending the local host name is a valid realm name. The resulting corner case could merit a check that the suggested realm is distinct from the fully qualifies host, and if not, simply strip the host name from the returned string before it is used in an application. One reason for sticking with the present behaviour, is that some systems respond with a non-qualified host name as reply from gethostname().

#### Returns

Returns a guessed realm for the running host, containing a string that has to be deallocated with free() by the caller.

### shishi_realm_default ()

```
const char~*
shishi_realm_default (Shishi *handle);
```

Determines name of default realm, i.e., the name of whatever realm the library will use whenever an explicit realm is not stated during a library call.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |

#### Returns

Returns the default realm in use by the library. Not a copy, so do not modify or deallocate the returned string.

### shishi_realm_default_set ()

```
void
shishi_realm_default_set (Shishi *handle,
                          const char *realm);
```

Sets the default realm used by the library; or, with *realm* set to NULL, resets the library realm setting to that name selected by configuration for default value.

The string is copied into the library, so you can dispose of the content in *realm* immediately after calling this function.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| realm | String stating a new default realm name, or NULL. |

### shishi_realm_for_server_file ()

```
char~*
shishi_realm_for_server_file (Shishi *handle,
                              char *server);
```

Finds the realm applicable to a host *server* , using the standard configuration file.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| server | Hostname to determine realm for. |

#### Returns

Returns realm for host, or NULL if not known.

### shishi_realm_for_server_dns ()

```
char~*
shishi_realm_for_server_dns (Shishi *handle,
                             char *server);
```

Finds the realm for a host *server* using DNS lookup, as is prescribed in "draft-ietf-krb-wg-krb-dns-locate-03.txt".

Since DNS lookup can be spoofed, relying on the realm information may result in a redirection attack. In a single-realm scenario, this only achieves a denial of service, but with trust across multiple realms the attack may redirect you to a compromised realm. For this reason, Shishi prints a warning, suggesting that the user should instead add a proper 'server-realm' configuration token.

To illustrate the DNS information used, here is an extract from a zone file for the domain ASDF.COM:

_kerberos.asdf.com. IN TXT "ASDF.COM" _kerberos.mrkserver.asdf.com. IN TXT "MARKETING.ASDF.COM" _kerberos.salesserve
IN TXT "SALES.ASDF.COM"

Let us suppose that in this case, a client wishes to use a service on the host "foo.asdf.com". It would first query for

_kerberos.foo.asdf.com. IN TXT

Finding no match, it would then query for

_kerberos.asdf.com. IN TXT

With the resource records stated above, the latter query returns a positive answer.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| server | Hostname to find realm for. | |

#### Returns

Returns realm for the indicated host, or NULL if no relevant TXT record could be found.

### shishi_realm_for_server ()

```
char~*
shishi_realm_for_server (Shishi *handle,
                         char *server);
```

Finds a realm for the host `server` , using various methods.

Currently this includes static configuration files, using the library call shishi_realm_for_server_file(), and DNS lookup using shishi_realm_for_server_dns(). They are attempted in the stated order. See the documentation of either function for more information.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| server | Hostname to find realm for. | |

#### Returns

Returns realm for the indicated host, or NULL if nothing is known about `server` .

### shishi_principal_default_guess ()

```
char~*
shishi_principal_default_guess (void);
```

Guesses the principal name for the user, looking at environment variables SHISHI_USER, USER and LOGNAME, or if that fails, returns the string "user".

#### Returns

Returns guessed default principal for user as a string that has to be deallocated by the caller with free().

### shishi_principal_default ()

```
const char~*
shishi_principal_default (Shishi *handle);
```

The default principal name is the name in the environment variable USER, or LOGNAME for some systems, but it can be overridden by specifying the environment variable SHISHI_USER.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |

#### Returns

Returns the default principal name used by the library. (Not a copy of it, so don't modify or deallocate it.)

### shishi_principal_default_set ()

```
void
shishi_principal_default_set (Shishi *handle,
                              const char *principal);
```

Set the default principal used by the library. The string is copied into the library, so you can dispose of the variable immediately after calling this function.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| principal | string with new default principal name, or NULL to reset to default. |

### shishi_principal_name ()

```
int
shishi_principal_name (Shishi *handle,
                       Shishi_asn1 namenode,
                       const char *namefield,
                       char **out,
                       size_t *outlen);
```

Represent principal name in ASN.1 structure as null-terminated string. The string is allocated by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `outlen` does not include the terminating null.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| namenode | ASN.1 structure with principal in *namefield* . | |
| namefield | name of field in *namenode* containing principal name. | |
| out | pointer to newly allocated, null terminated, string containing principal name. May be NULL (to only populate *outlen* ). | |
| outlen | pointer to length of *out* on output, excluding terminating null. May be NULL (to only populate *out* ). | |

### Returns

Returns SHISHI_OK if successful.

### shishi_principal_name_realm ()

```
int
shishi_principal_name_realm (Shishi *handle,
                             Shishi_asn1 namenode,
                             const char *namefield,
                             Shishi_asn1 realmnode,
                             const char *realmfield,
                             char **out,
                             size_t *outlen);
```

Represent principal name and realm in ASN.1 structure as null-terminated string. The string is allocated by this function. It is the responsibility of the caller to deallocate it. Note that the output length *outlen* does not include the terminating null character.

### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| namenode | ASN.1 structure with principal name in *namefield* . | |
| namefield | name of field in *namenode* containing principal name. | |
| realmnode | ASN.1 structure with principal realm in *realmfield* . | |
| realmfield | name of field in *realmnode* containing principal realm. | |
| out | pointer to newly allocated null terminated string containing principal name. May be NULL (to only populate *outlen* ). | |

| | |
|---|---|
| outlen | pointer to length of *out* on output, excluding terminating null. May be NULL (to only populate *out* ). |

### Returns

Returns SHISHI_OK if successful.

### shishi_principal_name_set ()

```
int
shishi_principal_name_set (Shishi *handle,
                           Shishi_asn1 namenode,
                           const char *namefield,
                           Shishi_name_type name_type,
                           const char *name[]);
```

Set the given principal name field to the given name.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| namenode | ASN.1 structure with principal in *namefield* . |
| namefield | name of field in *namenode* containing principal name. |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. |
| name | null-terminated input array with principal name. |

### Returns

Returns SHISHI_OK if successful.

### shishi_principal_set ()

```
int
shishi_principal_set (Shishi *handle,
                      Shishi_asn1 namenode,
                      const char *namefield,
                      const char *name);
```

Set principal name field in an ASN.1 structure to the given name.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| namenode | ASN.1 structure with principal in *namefield* . |
| namefield | name of field in *namenode* containing principal name. |
| name | null-terminated string with principal name in RFC 1964 form. |

### Returns

Returns SHISHI_OK if successful.

### shishi_parse_name ()

```
int
shishi_parse_name (Shishi *handle,
                   const char *name,
                   char **principal,
                   char **realm);
```

Split principal name (e.g., "simon@JOSEFSSON.ORG") into two newly allocated strings, the *principal* ("simon"), and the *realm* ("JOSEFSSON.ORG"). If there is no realm part in *name* , *realm* is set to NULL.

### Parameters

| | |
|---|---|
| handle | Shishi library handle created by shishi_init(). |
| name | input principal name string, e.g. imap/-mail.gnu.org@GNU.ORG. |
| principal | newly allocated output string with principal name. |
| realm | newly allocated output string with realm name. |

### Returns

Returns SHISHI_INVALID_PRINCIPAL_NAME if *name* is NULL or ends with the escape character "\", and SHISHI_OK if successful.

### shishi_derive_default_salt ()

```
int
shishi_derive_default_salt (Shishi *handle,
                            const char *name,
                            char **salt);
```

Derive the default salt from a principal. The default salt is the concatenation of the decoded realm and the principal.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| name | principal name of user. |
| salt | output variable with newly allocated salt string. |

**Returns**

Return SHISHI_OK if successful.

### shishi_server_for_local_service ()

```
char~*
shishi_server_for_local_service (Shishi *handle,
                                 const char *service);
```

Construct a service principal (e.g., "imap/yxa.extuno.com") based on supplied service name (i.e., "imap") and the system's hostname as returned by hostname() (i.e., "yxa.extundo.com"). The string must be deallocated by the caller.

**Parameters**

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| service | null terminated string with name of service, e.g., "host". |

**Returns**

Return newly allocated service name string.

### shishi_ticket ()

```
Shishi_asn1
shishi_ticket (Shishi *handle);
```

This function creates a new ASN.1 Ticket, populated with some default values.

**Parameters**

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

**Returns**

Returns the ticket or NULL on failure.

### shishi_ticket_server ()

```
int
shishi_ticket_server (Shishi *handle,
                      Shishi_asn1 ticket,
                      char **server,
                      size_t *serverlen);
```

Represent server principal name in Ticket as zero-terminated string. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `serverlen` does not include the terminating zero.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| ticket | ASN.1 Ticket variable to get server name from. | |
| server | pointer to newly allocated zero terminated string containing principal name. May be NULL (to only populate `serverlen` ). | |
| serverlen | pointer to length of `server` on output, excluding terminating zero. May be NULL (to only populate `server` ). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ticket_sname_set ()

```
int
shishi_ticket_sname_set (Shishi *handle,
                         Shishi_asn1 ticket,
                         Shishi_name_type name_type,
                         char *sname[]);
```

Set the server name field in the Ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ticket | Ticket variable to set server name field in. | |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. | |
| sname | input array with principal name. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ticket_srealmserver_set ()

```
int
shishi_ticket_srealmserver_set (Shishi *handle,
                                Shishi_asn1 ticket,
                                const char *realm,
                                const char *server);
```

### shishi_ticket_set_server ()

```
int
shishi_ticket_set_server (Shishi *handle,
                          Shishi_asn1 ticket,
                          const char *server);
```

### shishi_ticket_realm_get ()

```
int
shishi_ticket_realm_get (Shishi *handle,
                         Shishi_asn1 ticket,
                         char **realm,
                         size_t *realmlen);
```

Extract realm from ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ticket | input variable with ticket info. | |
| realm | output array with newly allocated name of realm in ticket. | |
| realmlen | size of output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ticket_realm_set ()

```
int
shishi_ticket_realm_set (Shishi *handle,
                         Shishi_asn1 ticket,
                         const char *realm);
```

Set the realm field in the Ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ticket | input variable with ticket info. | |
| realm | input array with name of realm. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ticket_get_enc_part_etype ()

```
int
shishi_ticket_get_enc_part_etype (Shishi *handle,
                                  Shishi_asn1 ticket,
                                  int32_t *etype);
```

Extract Ticket.enc-part.etype.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ticket | Ticket variable to get value from. | |
| etype | output variable that holds the value. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ticket_set_enc_part ()

```
int
shishi_ticket_set_enc_part (Shishi *handle,
                            Shishi_asn1 ticket,
                            int32_t etype,
                            uint32_t kvno,
                            const char *buf,
                            size_t buflen);
```

Set the encrypted enc-part field in the Ticket. The encrypted data is usually created by calling shishi_encrypt() on the DER encoded enc-part. To save time, you may want to use shishi_ticket_add_enc_part() instead, which calculates the encrypted data and calls this function in one step.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ticket | Ticket to add enc-part field to. | |

| | | |
|---|---|---|
| etype | encryption type used to encrypt enc-part. | |
| kvno | key version number. | |
| buf | input array with encrypted enc-part. | |
| buflen | size of input array with encrypted enc-part. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ticket_add_enc_part ()

```
int
shishi_ticket_add_enc_part (Shishi *handle,
                            Shishi_asn1 ticket,
                            Shishi_key *key,
                            Shishi_asn1 encticketpart);
```

Encrypts DER encoded EncTicketPart using key and stores it in the Ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ticket | Ticket to add enc-part field to. | |
| key | key used to encrypt enc-part. | |
| encticketpart | EncTicketPart to add. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ticket_decrypt ()

```
int
shishi_ticket_decrypt (Shishi *handle,
                       Shishi_asn1 ticket,
                       Shishi_key *key,
                       Shishi_asn1 *encticketpart);
```

### shishi_tkt_ticket ()

```
Shishi_asn1
shishi_tkt_ticket (Shishi_tkt *tkt);
```

Get ASN.1 Ticket structure from ticket.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns actual ticket.

### shishi_tkt_ticket_set ()

```
void
shishi_tkt_ticket_set (Shishi_tkt *tkt,
                       Shishi_asn1 ticket);
```

Set the ASN.1 Ticket in the Ticket.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| ticket | ASN.1 Ticket to store in ticket. |

### shishi_tkt_kdcrep ()

```
Shishi_asn1
shishi_tkt_kdcrep (Shishi_tkt *tkt);
```

Get ASN.1 KDCRep structure from ticket.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns KDC-REP information.

### shishi_tkt_enckdcreppart ()

```
Shishi_asn1
shishi_tkt_enckdcreppart (Shishi_tkt *tkt);
```

Get ASN.1 EncKDCRepPart structure from ticket.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns auxiliary ticket information.

### shishi_tkt_enckdcreppart_set ()

```
void
shishi_tkt_enckdcreppart_set (Shishi_tkt *tkt,
                              Shishi_asn1 enckdcreppart);
```

Set the EncKDCRepPart in the Ticket.

#### Parameters

| | | |
|---|---|---|
| tkt | structure that holds information about Ticket exchange | |
| enckdcreppart | EncKDCRepPart to store in Ticket. | |

### shishi_tkt_encticketpart ()

```
Shishi_asn1
shishi_tkt_encticketpart (Shishi_tkt *tkt);
```

Get ASN.1 EncTicketPart structure from ticket.

#### Parameters

| | | |
|---|---|---|
| tkt | input variable with ticket info. | |

#### Returns

Returns EncTicketPart information.

### shishi_tkt_encticketpart_set ()

```
void
shishi_tkt_encticketpart_set (Shishi_tkt *tkt,
                              Shishi_asn1 encticketpart);
```

Set the EncTicketPart in the Ticket.

#### Parameters

| | | |
|---|---|---|
| tkt | input variable with ticket info. | |
| encticketpart | encticketpart to store in ticket. | |

### shishi_tkt_key ()

```
Shishi_key~*
shishi_tkt_key (Shishi_tkt *tkt);
```

Get key used in ticket, by looking first in EncKDCRepPart and then in EncTicketPart. If key is already populated, it is not extracted again.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns key extracted from EncKDCRepPart or EncTicketPart.

### shishi_tkt_key_set ()

```
int
shishi_tkt_key_set (Shishi_tkt *tkt,
                    Shishi_key *key);
```

Set the key in the EncTicketPart.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| key | key to store in ticket. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt ()

```
int
shishi_tkt (Shishi *handle,
            Shishi_tkt **tkt);
```

Create a new ticket handle.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| tkt | output variable with newly allocated ticket. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt2 ()

```
Shishi_tkt~*
shishi_tkt2 (Shishi *handle,
             Shishi_asn1 ticket,
             Shishi_asn1 enckdcreppart,
             Shishi_asn1 kdcrep);
```

Create a new ticket handle.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ticket | input variable with ticket. | |
| enckdcreppart | input variable with auxiliary ticket information. | |
| kdcrep | input variable with KDC-REP ticket information. | |

#### Returns

Returns new ticket handle, or NULL on error.

### shishi_tkt_pretty_print ()

```
void
shishi_tkt_pretty_print (Shishi_tkt *tkt,
                         FILE *fh);
```

Print a human readable representation of a ticket to file handle.

#### Parameters

| | | |
|---|---|---|
| tkt | input variable with ticket info. | |
| fh | file handle open for writing. | |

### shishi_tkt_realm ()

```
int
shishi_tkt_realm (Shishi_tkt *tkt,
                  char **realm,
                  size_t *realmlen);
```

Extract realm of server in ticket.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| realm | pointer to newly allocated character array with realm name. |
| realmlen | length of newly allocated character array with realm name. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt_client ()

```
int
shishi_tkt_client (Shishi_tkt *tkt,
                   char **client,
                   size_t *clientlen);
```

Represent client principal name in Ticket KDC-REP as zero-terminated string. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `clientlen` does not include the terminating zero.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| client | pointer to newly allocated zero terminated string containing principal name. May be NULL (to only populate `clientlen` ). |
| clientlen | pointer to length of `client` on output, excluding terminating zero. May be NULL (to only populate `client` ). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt_client_p ()

```
int
shishi_tkt_client_p (Shishi_tkt *tkt,
                     const char *client);
```

Determine if ticket is for specified client.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| client | client name of ticket. |

### Returns

Returns non-0 iff ticket is for specified client.

### shishi_tkt_clientrealm ()

```
int
shishi_tkt_clientrealm (Shishi_tkt *tkt,
                        char **client,
                        size_t *clientlen);
```

Convert cname and realm fields from AS-REQ to printable principal name format. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `clientlen` does not include the terminating zero.

### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| client | pointer to newly allocated zero terminated string containing principal name and realm. May be NULL (to only populate `clientlen` ). |
| clientlen | pointer to length of `client` on output, excluding terminating zero. May be NULL (to only populate `client` ). |

### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt_clientrealm_p ()

```
int
shishi_tkt_clientrealm_p (Shishi_tkt *tkt,
                          const char *client);
```

Determine if ticket is for specified client principal.

### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| client | principal name (client name and realm) of ticket. |

#### Returns

Returns non-0 iff ticket is for specified client principal.

### shishi_tkt_clientrealm_set ()

```
int
shishi_tkt_clientrealm_set (Shishi_tkt *tkt,
                            const char *realm,
                            const char *client);
```

### shishi_tkt_serverrealm_set ()

```
int
shishi_tkt_serverrealm_set (Shishi_tkt *tkt,
                            const char *realm,
                            const char *server);
```

### shishi_tkt_build ()

```
int
shishi_tkt_build (Shishi_tkt *tkt,
                  Shishi_key *key);
```

### shishi_tkt_lastreq ()

```
int
shishi_tkt_lastreq (Shishi_tkt *tkt,
                    char **lrtime,
                    size_t *lrtimelen,
                    int32_t lrtype);
```

### shishi_tkt_lastreqc ()

```
time_t
shishi_tkt_lastreqc (Shishi_tkt *tkt,
                     Shishi_lrtype lrtype);
```

Extract C time corresponding to given lastreq type field in the ticket.

#### Parameters

| | | |
|---|---|---|
| tkt | input variable with ticket info. | |
| lrtype | lastreq type to extract, see Shishi_lrtype. E.g., SHISHI_LRTYPE_LAST_REQUEST. | |

#### Returns

Returns C time interpretation of the specified lastreq field, or (time_t) -1.

### shishi_tkt_lastreq_pretty_print ()

```
void
shishi_tkt_lastreq_pretty_print (Shishi_tkt *tkt,
                                 FILE *fh);
```

Print a human readable representation of the various lastreq fields in the ticket (really EncKDCRepPart).

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| fh | file handle open for writing. |

### shishi_tkt_authtime ()

```
int
shishi_tkt_authtime (Shishi_tkt *tkt,
                     char **authtime,
                     size_t *authtimelen);
```

### shishi_tkt_authctime ()

```
time_t
shishi_tkt_authctime (Shishi_tkt *tkt);
```

Extract C time corresponding to the authtime field. The field holds the time when the original authentication took place that later resulted in this ticket.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns C time interpretation of the endtime in ticket.

### shishi_tkt_starttime ()

```
int
shishi_tkt_starttime (Shishi_tkt *tkt,
                      char **starttime,
                      size_t *starttimelen);
```

### shishi_tkt_startctime ()

```
time_t
shishi_tkt_startctime (Shishi_tkt *tkt);
```

Extract C time corresponding to the starttime field. The field holds the time where the ticket start to be valid (typically in the past).

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns C time interpretation of the endtime in ticket.

### shishi_tkt_endtime ()

```
int
shishi_tkt_endtime (Shishi_tkt *tkt,
                    char **endtime,
                    size_t *endtimelen);
```

### shishi_tkt_endctime ()

```
time_t
shishi_tkt_endctime (Shishi_tkt *tkt);
```

Extract C time corresponding to the endtime field. The field holds the time where the ticket stop being valid.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns C time interpretation of the endtime in ticket.

### shishi_tkt_renew_till ()

```
int
shishi_tkt_renew_till (Shishi_tkt *tkt,
                       char **renewtilltime,
                       size_t *renewtilllen);
```

### shishi_tkt_renew_tillc ()

```
time_t
shishi_tkt_renew_tillc (Shishi_tkt *tkt);
```

Extract C time corresponding to the renew-till field. The field holds the time where the ticket stop being valid for renewal.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns C time interpretation of the renew-till in ticket.

### shishi_tkt_keytype ()

```
int
shishi_tkt_keytype (Shishi_tkt *tkt,
                    int32_t *etype);
```

Extract encryption type of key in ticket (really EncKDCRepPart).

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| etype | pointer to encryption type that is set, see Shishi_etype. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt_keytype_fast ()

```
int32_t
shishi_tkt_keytype_fast (Shishi_tkt *tkt);
```

Extract encryption type of key in ticket (really EncKDCRepPart).

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns encryption type of session key in ticket (really EncKDCRepPart), or -1 on error.

### shishi_tkt_keytype_p ()

```
int
shishi_tkt_keytype_p (Shishi_tkt *tkt,
                      int32_t etype);
```

Determine if key in ticket (really EncKDCRepPart) is of specified key type (really encryption type).

#### Parameters

| | | |
|---|---|---|
| tkt | input variable with ticket info. | |
| etype | encryption type, see Shishi_etype. | |

#### Returns

Returns non-0 iff key in ticket is of specified encryption type.

### shishi_tkt_server ()

```
int
shishi_tkt_server (Shishi_tkt *tkt,
                   char **server,
                   size_t *serverlen);
```

Represent server principal name in Ticket as zero-terminated string. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `serverlen` does not include the terminating zero.

#### Parameters

| | | |
|---|---|---|
| tkt | input variable with ticket info. | |
| server | pointer to newly allocated zero terminated string containing principal name. May be NULL (to only populate `serverlen` ). | |
| serverlen | pointer to length of `server` on output, excluding terminating zero. May be NULL (to only populate `server` ). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt_server_p ()

```
int
shishi_tkt_server_p (Shishi_tkt *tkt,
                     const char *server);
```

Determine if ticket is for specified server.

#### Parameters

| | | |
|---|---|---|
| tkt | input variable with ticket info. | |
| server | server name of ticket. | |

#### Returns

Returns non-0 iff ticket is for specified server.

### shishi_tkt_valid_at_time_p ()

```
int
shishi_tkt_valid_at_time_p (Shishi_tkt *tkt,
                            time_t now);
```

Determine if ticket is valid at a specific point in time.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| now | time to check for. |

#### Returns

Returns non-0 iff ticket is valid (not expired and after starttime) at specified time.

### shishi_tkt_valid_now_p ()

```
int
shishi_tkt_valid_now_p (Shishi_tkt *tkt);
```

Determine if ticket is valid now.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns 0 iff ticket is invalid (expired or not yet valid).

### shishi_tkt_expired_p ()

```
int
shishi_tkt_expired_p (Shishi_tkt *tkt);
```

Determine if ticket has expired (i.e., endtime is in the past).

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns 0 iff ticket has expired.

### shishi_tkt_decrypt ()

```
int
shishi_tkt_decrypt (Shishi_tkt *tkt,
                    Shishi_key *key);
```

### shishi_tkt_done ()

```
void
shishi_tkt_done (Shishi_tkt *tkt);
```

Deallocate resources associated with ticket. The ticket must not be used again after this call.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

### shishi_tkt_flags ()

```
int
shishi_tkt_flags (Shishi_tkt *tkt,
                  uint32_t *flags);
```

Extract flags in ticket (i.e., EncKDCRepPart).

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| flags | pointer to output integer with flags. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt_flags_set ()

```
int
shishi_tkt_flags_set (Shishi_tkt *tkt,
                      uint32_t flags);
```

Set flags in ticket, i.e., both EncTicketPart and EncKDCRepPart. Note that this reset any already existing flags.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| flags | integer with flags to store in ticket. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt_flags_add ()

```
int
shishi_tkt_flags_add (Shishi_tkt *tkt,
                      uint32_t flag);
```

Add ticket flags to Ticket and EncKDCRepPart. This preserves all existing options.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |
| flag | integer with flags to store in ticket. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkt_forwardable_p ()

```
int
shishi_tkt_forwardable_p (Shishi_tkt *tkt);
```

Determine if ticket is forwardable.

The FORWARDABLE flag in a ticket is normally only interpreted by the ticket-granting service. It can be ignored by application servers. The FORWARDABLE flag has an interpretation similar to that of the PROXIABLE flag, except ticket-granting tickets may also be issued with different network addresses. This flag is reset by default, but users MAY request that it be set by setting the FORWARDABLE option in the AS request when they request their initial ticket-granting ticket.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff forwardable flag is set in ticket.

### shishi_tkt_forwarded_p ()

```
int
```

```
shishi_tkt_forwarded_p (Shishi_tkt *tkt);
```

Determine if ticket is forwarded.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff forwarded flag is set in ticket.

### shishi_tkt_proxiable_p ()

```
int
shishi_tkt_proxiable_p (Shishi_tkt *tkt);
```

Determine if ticket is proxiable.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff proxiable flag is set in ticket.

### shishi_tkt_proxy_p ()

```
int
shishi_tkt_proxy_p (Shishi_tkt *tkt);
```

Determine if ticket is proxy ticket.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

### Returns

Returns non-0 iff proxy flag is set in ticket.

### shishi_tkt_may_postdate_p ()

```
int
shishi_tkt_may_postdate_p (Shishi_tkt *tkt);
```

Determine if ticket may be used to grant postdated tickets.

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

### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

### Returns

Returns non-0 iff may-postdate flag is set in ticket.

### shishi_tkt_postdated_p ()

```
int
shishi_tkt_postdated_p (Shishi_tkt *tkt);
```

Determine if ticket is postdated.

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

### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

### Returns

Returns non-0 iff postdated flag is set in ticket.

### shishi_tkt_invalid_p ()

```
int
shishi_tkt_invalid_p (Shishi_tkt *tkt);
```

Determine if ticket is invalid.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff invalid flag is set in ticket.

### shishi_tkt_renewable_p ()

```
int
shishi_tkt_renewable_p (Shishi_tkt *tkt);
```

Determine if ticket is renewable.

The RENEWABLE flag in a ticket is normally only interpreted by the ticket-granting service (discussed below in section 3.3). It can usually be ignored by application servers. However, some particularly careful application servers MAY disallow renewable tickets.

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff renewable flag is set in ticket.

### shishi_tkt_initial_p ()

```
int
shishi_tkt_initial_p (Shishi_tkt *tkt);
```

Determine if ticket was issued using AS exchange.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff initial flag is set in ticket.

### shishi_tkt_pre_authent_p ()

```
int
shishi_tkt_pre_authent_p (Shishi_tkt *tkt);
```

Determine if ticket was pre-authenticated.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff pre-authent flag is set in ticket.

### shishi_tkt_hw_authent_p ()

```
int
shishi_tkt_hw_authent_p (Shishi_tkt *tkt);
```

Determine if ticket is authenticated using a hardware token.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff hw-authent flag is set in ticket.

### shishi_tkt_transited_policy_checked_p ()

```
int
shishi_tkt_transited_policy_checked_p (Shishi_tkt *tkt);
```

Determine if ticket has been policy checked for transit.

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

While the end server ultimately decides whether authentication is valid, the KDC for the end server's realm MAY apply a realm specific policy for validating the transited field and accepting credentials for cross-realm authentication. When the KDC applies such checks and accepts such cross-realm authentication it will set the TRANSITED-POLICY-CHECKED flag in the service tickets it issues based on the cross-realm TGT. A client MAY request that the KDCs not check the transited field by setting the DISABLE-TRANSITED-CHECK flag. KDCs are encouraged but not required to honor this flag.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff transited-policy-checked flag is set in ticket.

### shishi_tkt_ok_as_delegate_p ()

```
int
shishi_tkt_ok_as_delegate_p (Shishi_tkt *tkt);
```

Determine if ticket is ok as delegated ticket.

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

#### Parameters

| | |
|---|---|
| tkt | input variable with ticket info. |

#### Returns

Returns non-0 iff ok-as-delegate flag is set in ticket.

### shishi_tkts_default_file_guess ()

```
char~*
shishi_tkts_default_file_guess (Shishi *handle);
```

Guesses the default ticket filename; it is $SHISHI_TICKETS, $SHISHI_HOME/tickets, or $HOME/.shishi/tickets.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Returns default tkts filename as a string that has to be deallocated with free() by the caller.

### shishi_tkts_default_file ()

```
const char~*
shishi_tkts_default_file (Shishi *handle);
```

Get filename of default ticket set.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Returns the default ticket set filename used in the library. The string is not a copy, so don't modify or deallocate it.

### shishi_tkts_default_file_set ()

```
void
shishi_tkts_default_file_set (Shishi *handle,
                              const char *tktsfile);
```

Set the default ticket set filename used in the library. The string is copied into the library, so you can dispose of the variable immediately after calling this function.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| tktsfile | string with new default tkts file name, or NULL to reset to default. |

### shishi_tkts_default ()

```
Shishi_tkts~*
```

```
shishi_tkts_default (Shishi *handle);
```

Get the default ticket set for library handle.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Return the handle global ticket set.

### shishi_tkts_default_to_file ()

```
int
shishi_tkts_default_to_file (Shishi_tkts *tkts);
```

### shishi_tkts ()

```
int
shishi_tkts (Shishi *handle,
             Shishi_tkts **tkts);
```

Get a new ticket set handle.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| tkts | output pointer to newly allocated tkts handle. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_nth ()

```
Shishi_tkt~*
shishi_tkts_nth (Shishi_tkts *tkts,
                 int ticketno);
```

Get the n:th ticket in ticket set.

#### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| ticketno | integer indicating requested ticket in ticket set. |

#### Returns

Returns a ticket handle to the ticketno:th ticket in the ticket set, or NULL if ticket set is invalid or ticketno is out of bounds. The first ticket is ticketno 0, the second ticketno 1, and so on.

### shishi_tkts_size ()

```
int
shishi_tkts_size (Shishi_tkts *tkts);
```

Get size of ticket set.

#### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |

#### Returns

Returns number of tickets stored in ticket set.

### shishi_tkts_add ()

```
int
shishi_tkts_add (Shishi_tkts *tkts,
                 Shishi_tkt *tkt);
```

Add a ticket to the ticket set. Only the pointer is stored, so if you modify *tkt* , the ticket in the ticket set will also be modified.

#### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| tkt | ticket to be added to ticket set. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_new ()

```
int
shishi_tkts_new (Shishi_tkts *tkts,
                 Shishi_asn1 ticket,
                 Shishi_asn1 enckdcreppart,
                 Shishi_asn1 kdcrep);
```

Allocate a new ticket and add it to the ticket set.

Note that *ticket* , *enckdcreppart* and *kdcrep* are stored by reference, so you must not de-allocate them before the ticket is removed from the ticket set and de-allocated.

### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| ticket | input ticket variable. | |
| enckdcreppart | input ticket detail variable. | |
| kdcrep | input KDC-REP variable. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_remove ()

```
int
shishi_tkts_remove (Shishi_tkts *tkts,
                    int ticketno);
```

Remove a ticket, indexed by `ticketno` , in ticket set.

### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| ticketno | ticket number of ticket in the set to remove. The first ticket is ticket number 0. | |

### Returns

SHISHI_OK if successful or if `ticketno` larger than size of ticket set.

### shishi_tkts_expire ()

```
int
shishi_tkts_expire (Shishi_tkts *tkts);
```

Remove expired tickets from ticket set.

### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_print_for_service ()

```
int
```

```
shishi_tkts_print_for_service (Shishi_tkts *tkts,
                               FILE *fh,
                               const char *service);
```

Print description of tickets for specified service to file descriptor. If service is NULL, all tickets are printed.

#### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| fh | file descriptor to print to. | |
| service | service to limit tickets printed to, or NULL. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_print ()

```
int
shishi_tkts_print (Shishi_tkts *tkts,
                   FILE *fh);
```

Print description of all tickets to file descriptor.

#### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| fh | file descriptor to print to. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_write ()

```
int
shishi_tkts_write (Shishi_tkts *tkts,
                   FILE *fh);
```

Write tickets in set to file descriptor.

#### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| fh | file descriptor to write tickets to. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_to_file ()

```
int
shishi_tkts_to_file (Shishi_tkts *tkts,
                     const char *filename);
```

Write tickets in set to file.

#### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| filename | filename to write tickets to. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_read ()

```
int
shishi_tkts_read (Shishi_tkts *tkts,
                  FILE *fh);
```

Read tickets from file descriptor and add them to the ticket set.

#### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| fh | file descriptor to read from. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_from_file ()

```
int
shishi_tkts_from_file (Shishi_tkts *tkts,
                       const char *filename);
```

Read tickets from file and add them to the ticket set.

#### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| filename | filename to read tickets from. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_tkts_done ()

```
void
shishi_tkts_done (Shishi_tkts **tkts);
```

Deallocates all resources associated with ticket set. The ticket set handle must not be used in calls to other shishi_tkts_*() functions after this.

### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |

### shishi_tkt_match_p ()

```
int
shishi_tkt_match_p (Shishi_tkt *tkt,
                    Shishi_tkts_hint *hint);
```

Test if a ticket matches specified hints.

### Parameters

| | | |
|---|---|---|
| tkt | ticket to test hints on. | |
| hint | structure with characteristics of ticket to be found. | |

### Returns

Returns 0 iff ticket fails to match given criteria.

### shishi_tkts_find ()

```
Shishi_tkt~*
shishi_tkts_find (Shishi_tkts *tkts,
                  Shishi_tkts_hint *hint);
```

Search the ticketset sequentially (from ticket number 0 through all tickets in the set) for a ticket that fits the given characteristics. If a ticket is found, the hint->startpos field is updated to point to the next ticket in the set, so this function can be called repeatedly with the same hint argument in order to find all tickets matching a certain criterium. Note that if tickets are added to, or removed from, the ticketset during a query with the same hint argument, the hint->startpos field must be updated appropriately.

Here is how you would typically use this function:

Shishi_tkts_hint hint;

Shishi_tkt tkt;

memset(&hint, 0, sizeof(hint));

hint.server = "imap/mail.example.org";

tkt = shishi_tkts_find (shishi_tkts_default(handle), &hint);

if (!tkt)

printf("No ticket found...\n");

else

do_something_with_ticket (tkt);

#### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| hint | structure with characteristics of ticket to be found. | |

#### Returns

Returns a ticket if found, or NULL if no further matching tickets could be found.

### shishi_tkts_find_for_clientserver ()

```
Shishi_tkt~*
shishi_tkts_find_for_clientserver (Shishi_tkts *tkts,
                                   const char *client,
                                   const char *server);
```

Short-hand function for searching the ticket set for a ticket for the given client and server. See shishi_tkts_find().

#### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| client | client name to find ticket for. | |
| server | server name to find ticket for. | |

#### Returns

Returns a ticket if found, or NULL.

### shishi_tkts_find_for_server ()

```
Shishi_tkt~*
shishi_tkts_find_for_server (Shishi_tkts *tkts,
                             const char *server);
```

Short-hand function for searching the ticket set for a ticket for the given server using the default client principal. See shishi_tkts_find_for_
and shishi_tkts_find().

#### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| server | server name to find ticket for. |

#### Returns

Returns a ticket if found, or NULL.

### shishi_tkts_get ()

```
Shishi_tkt~*
shishi_tkts_get (Shishi_tkts *tkts,
                 Shishi_tkts_hint *hint);
```

Get a ticket matching given characteristics. This function first looks in the ticket set for a ticket, then tries to find a suitable TGT, possibly via an AS exchange, using shishi_tkts_get_tgt(), and then uses that TGT in a TGS exchange to get the ticket.

Currently this function does not implement cross realm logic.

#### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| hint | structure with characteristics of ticket to be found. |

#### Returns

Returns a ticket if found, or NULL if this function is unable to get the ticket.

### shishi_tkts_get_tgt ()

```
Shishi_tkt~*
shishi_tkts_get_tgt (Shishi_tkts *tkts,
                     Shishi_tkts_hint *hint);
```

Get a ticket granting ticket (TGT) suitable for acquiring ticket matching the hint. I.e., get a TGT for the server realm in the hint structure (hint->serverrealm), or the default realm if the serverrealm field is NULL. Can result in AS exchange.

Currently this function do not implement cross realm logic.

This function is used by shishi_tkts_get(), which is probably what you really want to use unless you have special needs.

#### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| hint | structure with characteristics of ticket to begot. |

### Returns

Returns a ticket granting ticket if successful, or NULL if this function is unable to acquire on.

### shishi_tkts_get_tgs ()

```
Shishi_tkt~*
shishi_tkts_get_tgs (Shishi_tkts *tkts,
                     Shishi_tkts_hint *hint,
                     Shishi_tkt *tgt);
```

Get a ticket via TGS exchange using specified ticket granting ticket.

This function is used by shishi_tkts_get(), which is probably what you really want to use unless you have special needs.

### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| hint | structure with characteristics of ticket to begot. |
| tgt | ticket granting ticket to use. |

### Returns

Returns a ticket if successful, or NULL if this function is unable to acquire on.

### shishi_tkts_get_for_clientserver ()

```
Shishi_tkt~*
shishi_tkts_get_for_clientserver (Shishi_tkts *tkts,
                                  const char *client,
                                  const char *server);
```

Short-hand function for getting a ticket for the given client and server. See shishi_tkts_get().

### Parameters

| | |
|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). |
| client | client name to get ticket for. |
| server | server name to get ticket for. |

#### Returns

Returns a ticket if found, or NULL.

### shishi_tkts_get_for_server ()

```
Shishi_tkt~*
shishi_tkts_get_for_server (Shishi_tkts *tkts,
                            const char *server);
```

Short-hand function for getting a ticket to the given server and for the default principal client. See shishi_tkts_get().

#### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| server | server name to get ticket for. | |

#### Returns

Returns a ticket if found, or NULL.

### shishi_tkts_get_for_localservicepasswd ()

```
Shishi_tkt~*
shishi_tkts_get_for_localservicepasswd
                               (Shishi_tkts *tkts,
                                const char *service,
                                const char *passwd);
```

Short-hand function for getting a ticket to the given local service, and for the default principal client. The latter's password is given as argument. See shishi_tkts_get().

#### Parameters

| | | |
|---|---|---|
| tkts | ticket set handle as allocated by shishi_tkts(). | |
| service | service name to get ticket for. | |
| passwd | password for the default client principal. | |

#### Returns

Returns a ticket if found, or NULL otherwise.

### shishi_tkts_default_ccache_guess ()

```
char~*
shishi_tkts_default_ccache_guess (Shishi *handle);
```

Guesses the default ccache ticket filename; it is the contents of the environment variable KRB5CCNAME or /tmp/krb5cc_UID where UID is the user's identity in decimal, as returned by getuid().

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Returns default ccache filename as a string that has to be deallocated with free() by the caller.

### shishi_tkts_default_ccache ()

```
const char~*
shishi_tkts_default_ccache (Shishi *handle);
```

Get filename of default ccache filename.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Returns the default ccache filename used in the library. The string is not a copy, so don't modify or deallocate it.

### shishi_tkts_default_ccache_set ()

```
void
shishi_tkts_default_ccache_set (Shishi *handle,
                                const char *ccache);
```

Set the default ccache filename used in the library. The string is copied into the library, so you can dispose of the variable immediately after calling this function.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| ccache | string with new default ccache filename, or NULL to reset to default. |

### shishi_tkts_add_ccache_mem ()

```
int
shishi_tkts_add_ccache_mem (Shishi *handle,
                            const char *data,
                            size_t len,
                              Shishi_tkts *tkts);
```

Read tickets from a ccache data structure, and add them to the ticket set.

The ccache format is proprietary, and this function support (at least) the 0x0504 format. See the section The Credential Cache Binary File Format in the Shishi manual for a description of the file format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| data | constant memory buffer with ccache of `len` size. | |
| len | size of memory buffer with ccache data. | |
| tkts | allocated key set to store tickets in. | |

#### Returns

Returns SHISHI_CCACHE_ERROR if the data does not represent a valid ccache structure, and SHISHI_OK on success.

### shishi_tkts_add_ccache_file ()

```
int
shishi_tkts_add_ccache_file (Shishi *handle,
                             const char *filename,
                             Shishi_tkts *tkts);
```

Read tickets from a ccache data structure, and add them to the ticket set.

The ccache format is proprietary, and this function support (at least) the 0x0504 format. See the section The Credential Cache Binary File Format in the Shishi manual for a description of the file format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| filename | name of file to read. | |
| tkts | allocated ticket set to store tickets in. | |

#### Returns

Returns SHISHI_IO_ERROR if the file cannot be read, SHISHI_CCACHE_ERROR if the data cannot be parsed as a valid ccache structure, and SHISHI_OK on success.

### shishi_tkts_from_ccache_mem ()

```
int
shishi_tkts_from_ccache_mem (Shishi *handle,
                             const char *data,
                             size_t len,
                             Shishi_tkts **outtkts);
```

Read tickets from a ccache data structure, and add them to the ticket set.

The ccache format is proprietary, and this function support (at least) the 0x0504 format. See the section The Credential Cache Binary File Format in the Shishi manual for a description of the file format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| data | constant memory buffer with ccache of `len` size. | |
| len | size of memory buffer with ccache data. | |
| outtkts | pointer to ticket set that will be allocated and populated, must be deallocated by caller on succes. | |

#### Returns

Returns SHISHI_CCACHE_ERROR if the data does not represent a valid ccache structure, and SHISHI_OK on success.

### shishi_tkts_from_ccache_file ()

```
int
shishi_tkts_from_ccache_file (Shishi *handle,
                              const char *filename,
                              Shishi_tkts **outtkts);
```

Read tickets from a ccache data structure, and add them to the ticket set.

The ccache format is proprietary, and this function support (at least) the 0x0504 format. See the section The Credential Cache Binary File Format in the Shishi manual for a description of the file format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| filename | name of file to read. | |
| outtkts | pointer to ticket set that will be allocated and populated, must be deallocated by caller on succes. | |

#### Returns

Returns SHISHI_IO_ERROR if the file cannot be read, SHISHI_CCACHE_ERROR if the data cannot be parsed as a valid ccache structure, and SHISHI_OK on success.

### shishi_enckdcreppart_print ()

```
int
shishi_enckdcreppart_print (Shishi *handle,
                            FILE *fh,
                              Shishi_asn1 enckdcreppart);
```

### shishi_enckdcreppart_save ()

```
int
shishi_enckdcreppart_save (Shishi *handle,
                           FILE *fh,
                           Shishi_asn1 enckdcreppart);
```

### shishi_enckdcreppart_parse ()

```
int
shishi_enckdcreppart_parse (Shishi *handle,
                            FILE *fh,
                            Shishi_asn1 *enckdcreppart);
```

### shishi_enckdcreppart_read ()

```
int
shishi_enckdcreppart_read (Shishi *handle,
                           FILE *fh,
                           Shishi_asn1 *enckdcreppart);
```

### shishi_ticket_save ()

```
int
shishi_ticket_save (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 ticket);
```

### shishi_ticket_print ()

```
int
shishi_ticket_print (Shishi *handle,
                     FILE *fh,
                     Shishi_asn1 ticket);
```

### shishi_kdc_print ()

```
int
shishi_kdc_print (Shishi *handle,
                  FILE *fh,
                  Shishi_asn1 asreq,
                  Shishi_asn1 asrep,
                  Shishi_asn1 encasreppart);
```

### shishi_ticket_parse ()

```
int
shishi_ticket_parse (Shishi *handle,
                     FILE *fh,
                     Shishi_asn1 *ticket);
```

### shishi_ticket_read ()

```
int
shishi_ticket_read (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 *ticket);
```

### shishi_etype_info_print ()

```
int
shishi_etype_info_print (Shishi *handle,
                         FILE *fh,
                         Shishi_asn1 etypeinfo);
```

### shishi_etype_info2_print ()

```
int
shishi_etype_info2_print (Shishi *handle,
                          FILE *fh,
                          Shishi_asn1 etypeinfo2);
```

### shishi_padata_print ()

```
int
shishi_padata_print (Shishi *handle,
                     FILE *fh,
                     Shishi_asn1 padata);
```

### shishi_methoddata_print ()

```
int
shishi_methoddata_print (Shishi *handle,
                         FILE *fh,
                         Shishi_asn1 methoddata);
```

### shishi_authenticator ()

```
Shishi_asn1
shishi_authenticator (Shishi *handle);
```

This function creates a new Authenticator, populated with some default values. It uses the current time as returned by the system for the ctime and cusec fields.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the authenticator or NULL on failure.

### shishi_authenticator_set_crealm ()

```
int
shishi_authenticator_set_crealm (Shishi *handle,
                                 Shishi_asn1 authenticator,
                                 const char *crealm);
```

Set realm field in authenticator to specified value.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |
| crealm | input array with realm. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_set_cname ()

```
int
shishi_authenticator_set_cname (Shishi *handle,
                                Shishi_asn1 authenticator,
                                Shishi_name_type name_type,
                                const char *cname[]);
```

Set principal field in authenticator to specified value.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. |
| cname | input array with principal name. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_client_set ()

```
int
shishi_authenticator_client_set (Shishi *handle,
                                 Shishi_asn1 authenticator,
                                 const char *client);
```

Set the client name field in the Authenticator.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | Authenticator to set client name field in. | |
| client | zero-terminated string with principal name on RFC 1964 form. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_ctime ()

```
int
shishi_authenticator_ctime (Shishi *handle,
                            Shishi_asn1 authenticator,
                            char **t);
```

Extract client time from Authenticator.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | Authenticator as allocated by shishi_authenticator(). | |
| t | newly allocated zero-terminated character array with client time. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_ctime_set ()

```
int
shishi_authenticator_ctime_set (Shishi *handle,
                                Shishi_asn1 authenticator,
                                const char *t);
```

Store client time in Authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | Authenticator as allocated by shishi_authenticator(). |
| t | string with generalized time value to store in Authenticator. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_cusec_get ()

```
int
shishi_authenticator_cusec_get (Shishi *handle,
                                Shishi_asn1 authenticator,
                                uint32_t *cusec);
```

Extract client microseconds field from Authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | Authenticator as allocated by shishi_authenticator(). |
| cusec | output integer with client microseconds field. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_cusec_set ()

```
int
shishi_authenticator_cusec_set (Shishi *handle,
                                Shishi_asn1 authenticator,
                                uint32_t cusec);
```

Set the cusec field in the Authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |
| cusec | client microseconds to set in authenticator, 0-999999. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_seqnumber_get ()

```
int
shishi_authenticator_seqnumber_get (Shishi *handle,
                                    Shishi_asn1 authenticator,
                                    uint32_t *seqnumber);
```

Extract sequence number field from Authenticator.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |
| seqnumber | output integer with sequence number field. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_seqnumber_remove ()

```
int
shishi_authenticator_seqnumber_remove (Shishi *handle,
                                       Shishi_asn1 authenticator);
```

Remove sequence number field in Authenticator.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_seqnumber_set ()

```
int
shishi_authenticator_seqnumber_set (Shishi *handle,
                                    Shishi_asn1 authenticator,
                                    uint32_t seqnumber);
```

Store sequence number field in Authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |
| seqnumber | integer with sequence number field to store in Authenticator. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_client ()

```
int
shishi_authenticator_client (Shishi *handle,
                             Shishi_asn1 authenticator,
                             char **client,
                             size_t *clientlen);
```

Represent client principal name in Authenticator as zero-terminated string. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `clientlen` does not include the terminating zero.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| authenticator | Authenticator variable to get client name from. |
| client | pointer to newly allocated zero terminated string containing principal name. May be NULL (to only populate `clientlen` ). |
| clientlen | pointer to length of `client` on output, excluding terminating zero. May be NULL (to only populate `client` ). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_clientrealm ()

```
int
shishi_authenticator_clientrealm (Shishi *handle,
                                  Shishi_asn1 authenticator,
                                  char **client,
                                  size_t *clientlen);
```

Convert cname and realm fields from Authenticator to printable principal name format. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length *clientlen* does not include the terminating zero.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| authenticator | Authenticator variable to get client name and realm from. | |
| client | pointer to newly allocated zero terminated string containing principal name and realm. May be NULL (to only populate *clientlen* ). | |
| clientlen | pointer to length of *client* on output, excluding terminating zero. May be NULL (to only populate *client* ). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_remove_cksum ()

```
int
shishi_authenticator_remove_cksum (Shishi *handle,
                                   Shishi_asn1 authenticator);
```

### shishi_authenticator_cksum ()

```
int
shishi_authenticator_cksum (Shishi *handle,
                            Shishi_asn1 authenticator,
                            int32_t *cksumtype,
                            char **cksum,
                            size_t *cksumlen);
```

Read checksum value from authenticator. *cksum* is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |
| cksumtype | output checksum type. | |

| | | |
|---|---|---|
| cksum | newly allocated output checksum data from authenticator. | |
| cksumlen | on output, actual size of allocated output checksum data buffer. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_set_cksum ()

```
int
shishi_authenticator_set_cksum (Shishi *handle,
                                Shishi_asn1 authenticator,
                                int cksumtype,
                                char *cksum,
                                size_t cksumlen);
```

Store checksum value in authenticator. A checksum is usually created by calling shishi_checksum() on some application specific data using the key from the ticket that is being used. To save time, you may want to use shishi_authenticator_add_cksum() instead, which calculates the checksum and calls this function in one step.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |
| cksumtype | input checksum type to store in authenticator. | |
| cksum | input checksum data to store in authenticator. | |
| cksumlen | size of input checksum data to store in authenticator. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_add_cksum ()

```
int
shishi_authenticator_add_cksum (Shishi *handle,
                                Shishi_asn1 authenticator,
                                Shishi_key *key,
                                int keyusage,
                                char *data,
                                size_t datalen);
```

Calculate checksum for data and store it in the authenticator.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |
| key | key to to use for encryption. |
| keyusage | cryptographic key usage value to use in encryption. |
| data | input array with data to calculate checksum on. |
| datalen | size of input array with data to calculate checksum on. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_add_cksum_type ()

```
int
shishi_authenticator_add_cksum_type (Shishi *handle,
                                     Shishi_asn1 authenticator,
                                     Shishi_key *key,
                                     int keyusage,
                                     int cksumtype,
                                     char *data,
                                     size_t datalen);
```

Calculate checksum for data and store it in the authenticator.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |
| key | key to to use for encryption. |
| keyusage | cryptographic key usage value to use in encryption. |
| cksumtype | checksum to type to calculate checksum. |
| data | input array with data to calculate checksum on. |
| datalen | size of input array with data to calculate checksum on. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_remove_subkey ()

```
int
shishi_authenticator_remove_subkey (Shishi *handle,
                                    Shishi_asn1 authenticator);
```

Remove subkey from the authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_subkey ()

```
Shishi_asn1
shishi_authenticator_subkey (Shishi *handle);
```

This function creates a new Authenticator, populated with some default values. It uses the current time as returned by the system for the ctime and cusec fields. It adds a random subkey.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the authenticator or NULL on failure.

### shishi_authenticator_get_subkey ()

```
int
shishi_authenticator_get_subkey (Shishi *handle,
                                 Shishi_asn1 authenticator,
                                 Shishi_key **subkey);
```

Read subkey value from authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |
| subkey | output newly allocated subkey from authenticator. |

#### Returns

Returns SHISHI_OK if successful or SHISHI_ASN1_NO_ELEMENT if subkey is not present.

### shishi_authenticator_set_subkey ()

```
int
shishi_authenticator_set_subkey (Shishi *handle,
                                 Shishi_asn1 authenticator,
                                 int32_t subkeytype,
                                 const char *subkey,
                                 size_t subkeylen);
```

Store subkey value in authenticator. A subkey is usually created by calling shishi_key_random() using the default encryption type of the key from the ticket that is being used. To save time, you may want to use shishi_authenticator_add_subkey() instead, which calculates the subkey and calls this function in one step.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |
| subkeytype | input subkey type to store in authenticator. | |
| subkey | input subkey data to store in authenticator. | |
| subkeylen | size of input subkey data to store in authenticator. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_add_random_subkey ()

```
int
shishi_authenticator_add_random_subkey
                               (Shishi *handle,
                                Shishi_asn1 authenticator);
```

Generate random subkey, of the default encryption type from configuration, and store it in the authenticator.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_add_random_subkey_etype ()

```
int
```

```
shishi_authenticator_add_random_subkey_etype
                               (Shishi *handle,
                                Shishi_asn1 authenticator,
                                int etype);
```

Generate random subkey of indicated encryption type, and store it in the authenticator.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |
| etype | encryption type of random key to generate. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_add_subkey ()

```
int
shishi_authenticator_add_subkey (Shishi *handle,
                                 Shishi_asn1 authenticator,
                                 Shishi_key *subkey);
```

Store subkey in the authenticator.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |
| subkey | subkey to add to authenticator. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_clear_authorizationdata ()

```
int
shishi_authenticator_clear_authorizationdata
                               (Shishi *handle,
                                Shishi_asn1 authenticator);
```

Remove the authorization-data field from Authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | Authenticator as allocated by shishi_authenticator(). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_add_authorizationdata ()

```
int
shishi_authenticator_add_authorizationdata
                               (Shishi *handle,
                                Shishi_asn1 authenticator,
                                int32_t adtype,
                                const char *addata,
                                size_t addatalen);
```

Add authorization data to authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | authenticator as allocated by shishi_authenticator(). |
| adtype | input authorization data type to add. |
| addata | input authorization data to add. |
| addatalen | size of input authorization data to add. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_authorizationdata ()

```
int
shishi_authenticator_authorizationdata
                               (Shishi *handle,
                                Shishi_asn1 authenticator,
                                int32_t *adtype,
                                char **addata,
                                size_t *addatalen,
                                size_t nth);
```

Extract n:th authorization data from authenticator. The first field is 1.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |
| adtype | output authorization data type. | |
| addata | newly allocated output authorization data. | |
| addatalen | on output, actual size of newly allocated authorization data. | |
| nth | element number of authorization-data to extract. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_read ()

```
int
shishi_authenticator_read (Shishi *handle,
                           FILE *fh,
                           Shishi_asn1 *authenticator);
```

Read DER encoded authenticator from file and populate given authenticator variable.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for reading. | |
| authenticator | output variable with newly allocated authenticator. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_parse ()

```
int
shishi_authenticator_parse (Shishi *handle,
                            FILE *fh,
                            Shishi_asn1 *authenticator);
```

Read ASCII armored DER encoded authenticator from file and populate given authenticator variable.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| authenticator | output variable with newly allocated authenticator. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_from_file ()

```
int
shishi_authenticator_from_file (Shishi *handle,
                                Shishi_asn1 *authenticator,
                                int filetype,
                                const char *filename);
```

Read Authenticator from file in specified TYPE.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| authenticator | output variable with newly allocated Authenticator. |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. |
| filename | input variable with filename to read from. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_print ()

```
int
shishi_authenticator_print (Shishi *handle,
                            FILE *fh,
                            Shishi_asn1 authenticator);
```

Print ASCII armored DER encoding of authenticator to file.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for writing. |
| authenticator | authenticator as allocated by shishi_authenticator(). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_to_file ()

```
int
shishi_authenticator_to_file (Shishi *handle,
                              Shishi_asn1 authenticator,
                              int filetype,
                              const char *filename);
```

Write Authenticator to file in specified TYPE. The file will be truncated if it exists.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| authenticator | Authenticator to save. | |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. | |
| filename | input variable with filename to write to. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_authenticator_save ()

```
int
shishi_authenticator_save (Shishi *handle,
                           FILE *fh,
                           Shishi_asn1 authenticator);
```

Save DER encoding of authenticator to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as ()

```
int
shishi_as (Shishi *handle,
           Shishi_as **as);
```

Allocate a new AS exchange variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| as | holds pointer to newly allocate Shishi_as structure. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as_done ()

```
void
shishi_as_done (Shishi_as *as);
```

Deallocate resources associated with AS exchange. This should be called by the application when it no longer need to utilize the AS exchange handle.

#### Parameters

| | |
|---|---|
| as | structure that holds information about AS exchange |

### shishi_as_req ()

```
Shishi_asn1
shishi_as_req (Shishi_as *as);
```

Get ASN.1 AS-REQ structure from AS exchange.

#### Parameters

| | |
|---|---|
| as | structure that holds information about AS exchange |

#### Returns

Returns the generated AS-REQ packet from the AS exchange, or NULL if not yet set or an error occured.

### shishi_as_req_build ()

```
int
shishi_as_req_build (Shishi_as *as);
```

Possibly remove unset fields (e.g., rtime).

#### Parameters

| | |
|---|---|
| as | structure that holds information about AS exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as_req_set ()

```
void
shishi_as_req_set (Shishi_as *as,
                   Shishi_asn1 asreq);
```

Set the AS-REQ in the AS exchange.

#### Parameters

| | |
|---|---|
| as | structure that holds information about AS exchange |
| asreq | asreq to store in AS. |

### shishi_as_req_der ()

```
int
shishi_as_req_der (Shishi_as *as,
                   char **out,
                   size_t *outlen);
```

DER encode AS-REQ. `out` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | |
|---|---|
| as | structure that holds information about AS exchange |
| out | output array with newly allocated DER encoding of AS-REQ. |
| outlen | length of output array with DER encoding of AS-REQ. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as_req_der_set ()

```
int
shishi_as_req_der_set (Shishi_as *as,
                       char *der,
                       size_t derlen);
```

DER decode AS-REQ and set it AS exchange. If decoding fails, the AS-REQ in the AS exchange remains.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |
| der | input array with DER encoded AP-REQ. | |
| derlen | length of input array with DER encoded AP-REQ. | |

#### Returns

Returns SHISHI_OK.

### shishi_as_rep ()

```
Shishi_asn1
shishi_as_rep (Shishi_as *as);
```

Get ASN.1 AS-REP structure from AS exchange.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |

#### Returns

Returns the received AS-REP packet from the AS exchange, or NULL if not yet set or an error occured.

### shishi_as_rep_set ()

```
void
shishi_as_rep_set (Shishi_as *as,
                   Shishi_asn1 asrep);
```

Set the AS-REP in the AS exchange.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |
| asrep | asrep to store in AS. | |

### shishi_as_rep_build ()

```
int
shishi_as_rep_build (Shishi_as *as,
                     Shishi_key *key);
```

Build AS-REP.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |
| key | user's key, used to encrypt the encrypted part of the AS-REP. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as_rep_der ()

```
int
shishi_as_rep_der (Shishi_as *as,
                   char **out,
                   size_t *outlen);
```

DER encode AS-REP. `out` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |
| out | output array with newly allocated DER encoding of AS-REP. | |
| outlen | length of output array with DER encoding of AS-REP. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as_rep_der_set ()

```
int
shishi_as_rep_der_set (Shishi_as *as,
                       char *der,
                       size_t derlen);
```

DER decode AS-REP and set it AS exchange. If decoding fails, the AS-REP in the AS exchange remains.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |
| der | input array with DER encoded AP-REP. | |
| derlen | length of input array with DER encoded AP-REP. | |

#### Returns

Returns SHISHI_OK.

### shishi_as_krberror ()

```
Shishi_asn1
shishi_as_krberror (Shishi_as *as);
```

Get ASN.1 KRB-ERROR structure from AS exchange.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |

#### Returns

Returns the received KRB-ERROR packet from the AS exchange, or NULL if not yet set or an error occured.

### shishi_as_krberror_der ()

```
int
shishi_as_krberror_der (Shishi_as *as,
                        char **out,
                        size_t *outlen);
```

DER encode KRB-ERROR. `out` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |
| out | output array with newly allocated DER encoding of KRB-ERROR. | |
| outlen | length of output array with DER encoding of KRB-ERROR. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as_krberror_set ()

```
void
shishi_as_krberror_set (Shishi_as *as,
                        Shishi_asn1 krberror);
```

Set the KRB-ERROR in the AS exchange.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |
| krberror | krberror to store in AS. | |

### shishi_as_tkt ()

```
Shishi_tkt~*
shishi_as_tkt (Shishi_as *as);
```

Get Ticket in AS exchange.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |

#### Returns

Returns the newly acquired tkt from the AS exchange, or NULL if not yet set or an error occured.

### shishi_as_tkt_set ()

```
void
shishi_as_tkt_set (Shishi_as *as,
                   Shishi_tkt *tkt);
```

Set the Tkt in the AS exchange.

#### Parameters

| | |
|---|---|
| as | structure that holds information about AS exchange |
| tkt | tkt to store in AS. |

### shishi_as_sendrecv ()

```
int
shishi_as_sendrecv (Shishi_as *as);
```

Send AS-REQ and receive AS-REP or KRB-ERROR. This is the initial authentication, usually used to acquire a Ticket Granting Ticket.

#### Parameters

| | |
|---|---|
| as | structure that holds information about AS exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as_sendrecv_hint ()

```
int
shishi_as_sendrecv_hint (Shishi_as *as,
                         Shishi_tkts_hint *hint);
```

Send AS-REQ and receive AS-REP or KRB-ERROR. This is the initial authentication, usually used to acquire a Ticket Granting Ticket. The *hint* structure can be used to set, e.g., parameters for TLS authentication.

#### Parameters

| | |
|---|---|
| as | structure that holds information about AS exchange |
| hint | additional parameters that modify connection behaviour, or NULL. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_as_rep_process ()

```
int
shishi_as_rep_process (Shishi_as *as,
                       Shishi_key *key,
                                const char *password);
```

Process new AS-REP and set ticket. The key is used to decrypt the AP-REP. If both key and password is NULL, the user is queried for it.

#### Parameters

| | | |
|---|---|---|
| as | structure that holds information about AS exchange | |
| key | user's key, used to encrypt the encrypted part of the AS-REP. | |
| password | user's password, used if key is NULL. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs ()

```
int
shishi_tgs (Shishi *handle,
            Shishi_tgs **tgs);
```

Allocate a new TGS exchange variable.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| tgs | holds pointer to newly allocate Shishi_tgs structure. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_done ()

```
void
shishi_tgs_done (Shishi_tgs *tgs);
```

Deallocate resources associated with TGS exchange. This should be called by the application when it no longer need to utilize the TGS exchange handle.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about AS exchange |

### shishi_tgs_tgtkt ()

```
Shishi_tkt~*
shishi_tgs_tgtkt (Shishi_tgs *tgs);
```

Get Ticket-granting-ticket from TGS exchange.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |

#### Returns

Returns the ticket-granting-ticket used in the TGS exchange, or NULL if not yet set or an error occured.

### shishi_tgs_tgtkt_set ()

```
void
shishi_tgs_tgtkt_set (Shishi_tgs *tgs,
                      Shishi_tkt *tgtkt);
```

Set the Ticket in the TGS exchange.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |
| tgtkt | ticket granting ticket to store in TGS. |

### shishi_tgs_ap ()

```
Shishi_ap~*
shishi_tgs_ap (Shishi_tgs *tgs);
```

Get the AP from TGS exchange.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |

#### Returns

Returns the AP exchange (part of TGS-REQ) from the TGS exchange, or NULL if not yet set or an error occured.

### shishi_tgs_req ()

```
Shishi_asn1
shishi_tgs_req (Shishi_tgs *tgs);
```

Get the TGS-REQ from TGS exchange.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |

#### Returns

Returns the generated TGS-REQ from the TGS exchange, or NULL if not yet set or an error occured.

### shishi_tgs_req_der ()

```
int
shishi_tgs_req_der (Shishi_tgs *tgs,
                    char **out,
                    size_t *outlen);
```

DER encode TGS-REQ. `out` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |
| out | output array with newly allocated DER encoding of TGS-REQ. | |
| outlen | length of output array with DER encoding of TGS-REQ. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_req_der_set ()

```
int
shishi_tgs_req_der_set (Shishi_tgs *tgs,
                        char *der,
                        size_t derlen);
```

DER decode TGS-REQ and set it TGS exchange. If decoding fails, the TGS-REQ in the TGS exchange remains.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |
| der | input array with DER encoded AP-REQ. | |
| derlen | length of input array with DER encoded AP-REQ. | |

#### Returns

Returns SHISHI_OK.

### shishi_tgs_req_set ()

```
void
shishi_tgs_req_set (Shishi_tgs *tgs,
                    Shishi_asn1 tgsreq);
```

Set the TGS-REQ in the TGS exchange.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |
| tgsreq | tgsreq to store in TGS. | |

### shishi_tgs_req_build ()

```
int
shishi_tgs_req_build (Shishi_tgs *tgs);
```

Checksum data in authenticator and add ticket and authenticator to TGS-REQ.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_req_process ()

```
int
```

```
shishi_tgs_req_process (Shishi_tgs *tgs);
```

Process new TGS-REQ and set ticket. The key to decrypt the TGS-REQ is taken from the EncKDCReqPart of the TGS tgticket.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_rep ()

```
Shishi_asn1
shishi_tgs_rep (Shishi_tgs *tgs);
```

Get TGS-REP from TGS exchange.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |

#### Returns

Returns the received TGS-REP from the TGS exchange, or NULL if not yet set or an error occured.

### shishi_tgs_rep_der ()

```
int
shishi_tgs_rep_der (Shishi_tgs *tgs,
                    char **out,
                    size_t *outlen);
```

DER encode TGS-REP. `out` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |
| out | output array with newly allocated DER encoding of TGS-REP. |
| outlen | length of output array with DER encoding of TGS-REP. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_rep_build ()

```
int
shishi_tgs_rep_build (Shishi_tgs *tgs,
                      int keyusage,
                      Shishi_key *key);
```

Build TGS-REP.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |
| keyusage | keyusage integer. | |
| key | user's key, used to encrypt the encrypted part of the TGS-REP. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_rep_process ()

```
int
shishi_tgs_rep_process (Shishi_tgs *tgs);
```

Process new TGS-REP and set ticket. The key to decrypt the TGS-REP is taken from the EncKDCRepPart of the TGS tgticket.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_krberror ()

```
Shishi_asn1
shishi_tgs_krberror (Shishi_tgs *tgs);
```

Get KRB-ERROR from TGS exchange.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |

#### Returns

Returns the received TGS-REP from the TGS exchange, or NULL if not yet set or an error occured.

### shishi_tgs_krberror_der ()

```
int
shishi_tgs_krberror_der (Shishi_tgs *tgs,
                         char **out,
                         size_t *outlen);
```

DER encode KRB-ERROR. *out* is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |
| out | output array with newly allocated DER encoding of KRB-ERROR. |
| outlen | length of output array with DER encoding of KRB-ERROR. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_krberror_set ()

```
void
shishi_tgs_krberror_set (Shishi_tgs *tgs,
                         Shishi_asn1 krberror);
```

Set the KRB-ERROR in the TGS exchange.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |
| krberror | krberror to store in TGS. |

### shishi_tgs_tkt ()

```
Shishi_tkt~*
shishi_tgs_tkt (Shishi_tgs *tgs);
```

Get Ticket from TGS exchange.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |

#### Returns

Returns the newly acquired ticket from the TGS exchange, or NULL if not yet set or an error occured.

### shishi_tgs_tkt_set ()

```
void
shishi_tgs_tkt_set (Shishi_tgs *tgs,
                    Shishi_tkt *tkt);
```

Set the Ticket in the TGS exchange.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |
| tkt | ticket to store in TGS. |

### shishi_tgs_sendrecv ()

```
int
shishi_tgs_sendrecv (Shishi_tgs *tgs);
```

Send TGS-REQ and receive TGS-REP or KRB-ERROR. This is the subsequent authentication, usually used to acquire server tickets.

#### Parameters

| | |
|---|---|
| tgs | structure that holds information about TGS exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_sendrecv_hint ()

```
int
shishi_tgs_sendrecv_hint (Shishi_tgs *tgs,
                          Shishi_tkts_hint *hint);
```

Send TGS-REQ and receive TGS-REP or KRB-ERROR. This is the subsequent authentication, usually used to acquire server tickets. The `hint` structure can be used to set, e.g., parameters for TLS authentication.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |
| hint | additional parameters that modify connection behaviour, or NULL. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_set_server ()

```
int
shishi_tgs_set_server (Shishi_tgs *tgs,
                       const char *server);
```

Set the server in the TGS-REQ.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |
| server | indicates the server to acquire ticket for. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_set_realm ()

```
int
shishi_tgs_set_realm (Shishi_tgs *tgs,
                      const char *realm);
```

Set the server in the TGS-REQ.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |
| realm | indicates the realm to acquire ticket for. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_tgs_set_realmserver ()

```
int
shishi_tgs_set_realmserver (Shishi_tgs *tgs,
                            const char *realm,
                            const char *server);
```

Set the realm and server in the TGS-REQ.

#### Parameters

| | | |
|---|---|---|
| tgs | structure that holds information about TGS exchange | |
| realm | indicates the realm to acquire ticket for. | |
| server | indicates the server to acquire ticket for. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq ()

```
int
shishi_kdcreq (Shishi *handle,
               char *realm,
               char *service,
               Shishi_asn1 *req);
```

### shishi_asreq ()

```
Shishi_asn1
shishi_asreq (Shishi *handle);
```

This function creates a new AS-REQ, populated with some default values.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |

#### Returns

Returns the AS-REQ or NULL on failure.

### shishi_asreq_rsc ()

```
Shishi_asn1
shishi_asreq_rsc (Shishi *handle,
                  char *realm,
                  char *server,
                  char *client);
```

### shishi_tgsreq ()

```
Shishi_asn1
shishi_tgsreq (Shishi *handle);
```

This function creates a new TGS-REQ, populated with some default values.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the TGS-REQ or NULL on failure.

### shishi_tgsreq_rst ()

```
Shishi_asn1
shishi_tgsreq_rst (Shishi *handle,
                   char *realm,
                   char *server,
                   Shishi_tkt *tkt);
```

### shishi_kdcreq_save ()

```
int
shishi_kdcreq_save (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 kdcreq);
```

Print DER encoding of KDC-REQ to file.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for writing. |
| kdcreq | KDC-REQ to save. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_print ()

```
int
shishi_kdcreq_print (Shishi *handle,
                     FILE *fh,
                     Shishi_asn1 kdcreq);
```

Print ASCII armored DER encoding of KDC-REQ to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| kdcreq | KDC-REQ to print. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_to_file ()

```
int
shishi_kdcreq_to_file (Shishi *handle,
                       Shishi_asn1 kdcreq,
                       int filetype,
                       const char *filename);
```

Write KDC-REQ to file in specified TYPE. The file will be truncated if it exists.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ to save. | |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. | |
| filename | input variable with filename to write to. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_parse ()

```
int
shishi_kdcreq_parse (Shishi *handle,
                     FILE *fh,
                     Shishi_asn1 *kdcreq);
```

Read ASCII armored DER encoded KDC-REQ from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| kdcreq | output variable with newly allocated KDC-REQ. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_read ()

```
int
shishi_kdcreq_read (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 *kdcreq);
```

Read DER encoded KDC-REQ from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| kdcreq | output variable with newly allocated KDC-REQ. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_from_file ()

```
int
shishi_kdcreq_from_file (Shishi *handle,
                         Shishi_asn1 *kdcreq,
                         int filetype,
                         const char *filename);
```

Read KDC-REQ from file in specified TYPE.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | output variable with newly allocated KDC-REQ. |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. |
| filename | input variable with filename to read from. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_asreq_clientrealm ()

```
int
shishi_asreq_clientrealm (Shishi *handle,
                          Shishi_asn1 asreq,
                          char **client,
                          size_t *clientlen);
```

Convert cname and realm fields from AS-REQ to printable principal name format. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `clientlen` does not include the terminating zero.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| asreq | AS-REQ variable to get client name and realm from. | |
| client | pointer to newly allocated zero terminated string containing principal name and realm. May be NULL (to only populate *clientlen* ). | |
| clientlen | pointer to length of *client* on output, excluding terminating zero. May be NULL (to only populate *client* ). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_nonce ()

```
int
shishi_kdcreq_nonce (Shishi *handle,
                     Shishi_asn1 kdcreq,
                     uint32_t *nonce);
```

### shishi_kdcreq_nonce_set ()

```
int
shishi_kdcreq_nonce_set (Shishi *handle,
                         Shishi_asn1 kdcreq,
                         uint32_t nonce);
```

Store nonce number field in KDC-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to set client name field in. |
| nonce | integer nonce to store in KDC-REQ. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_client ()

```
int
shishi_kdcreq_client (Shishi *handle,
                      Shishi_asn1 kdcreq,
                      char **client,
                      size_t *clientlen);
```

Represent client principal name in KDC-REQ as zero-terminated string. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `clientlen` does not include the terminating zero.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| kdcreq | KDC-REQ variable to get client name from. |
| client | pointer to newly allocated zero terminated string containing principal name. May be NULL (to only populate `clientlen` ). |
| clientlen | pointer to length of `client` on output, excluding terminating zero. May be NULL (to only populate `client` ). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_set_cname ()

```
int
shishi_kdcreq_set_cname (Shishi *handle,
                         Shishi_asn1 kdcreq,
                         Shishi_name_type name_type,
                         const char *principal);
```

Set the client name field in the KDC-REQ.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to set client name field in. | |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. | |
| principal | input array with principal name. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_server ()

```
int
shishi_kdcreq_server (Shishi *handle,
                      Shishi_asn1 kdcreq,
                      char **server,
                      size_t *serverlen);
```

Represent server principal name in KDC-REQ as zero-terminated string. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `serverlen` does not include the terminating zero.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| kdcreq | KDC-REQ variable to get server name from. | |
| server | pointer to newly allocated zero terminated string containing principal name. May be NULL (to only populate `serverlen` ). | |
| serverlen | pointer to length of `server` on output, excluding terminating zero. May be NULL (to only populate `server` ). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_set_sname ()

```
int
shishi_kdcreq_set_sname (Shishi *handle,
                         Shishi_asn1 kdcreq,
                         Shishi_name_type name_type,
                         const char *sname[]);
```

Set the server name field in the KDC-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to set server name field in. |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. |
| sname | input array with principal name. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_realm ()

```
int
shishi_kdcreq_realm (Shishi *handle,
                     Shishi_asn1 kdcreq,
                     char **realm,
                     size_t *realmlen);
```

Get realm field in KDC-REQ as zero-terminated string. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `realmlen` does not include the terminating zero.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| kdcreq | KDC-REQ variable to get client name from. |
| realm | pointer to newly allocated zero terminated string containing realm. May be NULL (to only populate `realmlen` ). |
| realmlen | pointer to length of `realm` on output, excluding terminating zero. May be NULL (to only populate `realmlen` ). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_realm_get ()

```
int
shishi_kdcreq_realm_get (Shishi *handle,
                         Shishi_asn1 kdcreq,
                         char **realm,
                         size_t *realmlen);
```

### shishi_kdcreq_set_realm ()

```
int
shishi_kdcreq_set_realm (Shishi *handle,
                         Shishi_asn1 kdcreq,
                         const char *realm);
```

Set the realm field in the KDC-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to set realm field in. |
| realm | input array with name of realm. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_set_server ()

```
int
shishi_kdcreq_set_server (Shishi *handle,
                          Shishi_asn1 req,
                          const char *service);
```

### shishi_kdcreq_set_realmserver ()

```
int
shishi_kdcreq_set_realmserver (Shishi *handle,
                               Shishi_asn1 req,
                               char *realm,
                               char *service);
```

### shishi_kdcreq_till ()

```
int
shishi_kdcreq_till (Shishi *handle,
                    Shishi_asn1 kdcreq,
                    char **till,
                    size_t *tilllen);
```

Get "till" field, i.e., "endtime", in KDC-REQ as a null-terminated string. The string is typically 15 characters long and is allocated by this function. It is the responsibility of the caller to deallocate it. Note that the output length `tilllen` does not include the terminating zero.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| kdcreq | KDC-REQ variable to get endtime from. | |
| till | pointer to newly allocated null terminated string containing "till" field with generalized time. May be passed as NULL to only populate *tillen* . | |
| tilllen | pointer to length of *till* for output, excluding the terminating null. Set to NULL, only *till* is populated. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_tillc ()

```
time_t
shishi_kdcreq_tillc (Shishi *handle,
                     Shishi_asn1 kdcreq);
```

Extract C time corresponding to the "till" field.

### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| kdcreq | KDC-REQ variable to get "till" field from. | |

### Returns

Returns the C time interpretation of the "till" field in KDC-REQ.

### shishi_kdcreq_etype ()

```
int
shishi_kdcreq_etype (Shishi *handle,
                     Shishi_asn1 kdcreq,
                     int32_t *etype,
                     int netype);
```

Return the netype:th encryption type from KDC-REQ. The first etype is number 1.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to get etype field from. |
| etype | output encryption type. |
| netype | element number to return. |

#### Returns

Returns SHISHI_OK iff etype successful set.

### shishi_kdcreq_set_etype ()

```
int
shishi_kdcreq_set_etype (Shishi *handle,
                         Shishi_asn1 kdcreq,
                         int32_t *etype,
                         int netype);
```

Set the list of supported or wanted encryption types in the request. The list should be sorted in priority order.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to set etype field in. |
| etype | input array with encryption types. |
| netype | number of elements in input array with encryption types. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_options ()

```
int
shishi_kdcreq_options (Shishi *handle,
                       Shishi_asn1 kdcreq,
                       uint32_t *flags);
```

Extract KDC-Options from KDC-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to get kdc-options field from. |
| flags | pointer to output integer with flags. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_forwardable_p ()

```
int
shishi_kdcreq_forwardable_p (Shishi *handle,
                             Shishi_asn1 kdcreq);
```

Determine if KDC-Option forwardable flag is set.

The FORWARDABLE option indicates that the ticket to be issued is to have its forwardable flag set. It may only be set on the initial request, or in a subsequent request if the ticket-granting ticket on which it is based is also forwardable.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to get kdc-options field from. | |

#### Returns

Returns non-0 iff forwardable flag is set in KDC-REQ.

### shishi_kdcreq_forwarded_p ()

```
int
shishi_kdcreq_forwarded_p (Shishi *handle,
                           Shishi_asn1 kdcreq);
```

Determine if KDC-Option forwarded flag is set.

The FORWARDED option is only specified in a request to the ticket-granting server and will only be honored if the ticket-granting ticket in the request has its FORWARDABLE bit set. This option indicates that this is a request for forwarding. The address(es) of the host from which the resulting ticket is to be valid are included in the addresses field of the request.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to get kdc-options field from. | |

#### Returns

Returns non-0 iff forwarded flag is set in KDC-REQ.

### shishi_kdcreq_proxiable_p ()

```
int
shishi_kdcreq_proxiable_p (Shishi *handle,
                              Shishi_asn1 kdcreq);
```

Determine if KDC-Option proxiable flag is set.

The PROXIABLE option indicates that the ticket to be issued is to have its proxiable flag set. It may only be set on the initial request, or in a subsequent request if the ticket-granting ticket on which it is based is also proxiable.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to get kdc-options field from. | |

#### Returns

Returns non-0 iff proxiable flag is set in KDC-REQ.

### shishi_kdcreq_proxy_p ()

```
int
shishi_kdcreq_proxy_p (Shishi *handle,
                       Shishi_asn1 kdcreq);
```

Determine if KDC-Option proxy flag is set.

The PROXY option indicates that this is a request for a proxy. This option will only be honored if the ticket-granting ticket in the request has its PROXIABLE bit set. The address(es) of the host from which the resulting ticket is to be valid are included in the addresses field of the request.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to get kdc-options field from. | |

#### Returns

Returns non-0 iff proxy flag is set in KDC-REQ.

### shishi_kdcreq_allow_postdate_p ()

```
int
shishi_kdcreq_allow_postdate_p (Shishi *handle,
                                Shishi_asn1 kdcreq);
```

Determine if KDC-Option allow-postdate flag is set.

The ALLOW-POSTDATE option indicates that the ticket to be issued is to have its MAY-POSTDATE flag set. It may only be set on the initial request, or in a subsequent request if the ticket-granting ticket on which it is based also has its MAY-POSTDATE flag set.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to get kdc-options field from. |

#### Returns

Returns non-0 iff allow-postdate flag is set in KDC-REQ.

### shishi_kdcreq_postdated_p ()

```
int
shishi_kdcreq_postdated_p (Shishi *handle,
                           Shishi_asn1 kdcreq);
```

Determine if KDC-Option postdated flag is set.

The POSTDATED option indicates that this is a request for a postdated ticket. This option will only be honored if the ticket-granting ticket on which it is based has its MAY-POSTDATE flag set. The resulting ticket will also have its INVALID flag set, and that flag may be reset by a subsequent request to the KDC after the starttime in the ticket has been reached.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to get kdc-options field from. |

#### Returns

Returns non-0 iff postdated flag is set in KDC-REQ.

### shishi_kdcreq_renewable_p ()

```
int
shishi_kdcreq_renewable_p (Shishi *handle,
                           Shishi_asn1 kdcreq);
```

Determine if KDC-Option renewable flag is set.

The RENEWABLE option indicates that the ticket to be issued is to have its RENEWABLE flag set. It may only be set on the initial request, or when the ticket-granting ticket on which the request is based is also renewable. If this option is requested, then the rtime field in the request contains the desired absolute expiration time for the ticket.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to get kdc-options field from. |

#### Returns

Returns non-0 iff renewable flag is set in KDC-REQ.

### shishi_kdcreq_disable_transited_check_p ()

```
int
shishi_kdcreq_disable_transited_check_p
                               (Shishi *handle,
                                Shishi_asn1 kdcreq);
```

Determine if KDC-Option disable-transited-check flag is set.

By default the KDC will check the transited field of a ticket-granting-ticket against the policy of the local realm before it will issue derivative tickets based on the ticket-granting ticket. If this flag is set in the request, checking of the transited field is disabled. Tickets issued without the performance of this check will be noted by the reset (0) value of the TRANSITED-POLICY-CHECKED flag, indicating to the application server that the tranisted field must be checked locally. KDCs are encouraged but not required to honor the DISABLE-TRANSITED-CHECK option.

This flag is new since RFC 1510

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to get kdc-options field from. | |

#### Returns

Returns non-0 iff disable-transited-check flag is set in KDC-REQ.

### shishi_kdcreq_renewable_ok_p ()

```
int
shishi_kdcreq_renewable_ok_p (Shishi *handle,
                              Shishi_asn1 kdcreq);
```

Determine if KDC-Option renewable-ok flag is set.

The RENEWABLE-OK option indicates that a renewable ticket will be acceptable if a ticket with the requested life cannot otherwise be provided. If a ticket with the requested life cannot be provided, then a renewable ticket may be issued with a renew-till equal to the requested endtime. The value of the renew-till field may still be limited by local limits, or limits selected by the individual principal or server.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to get kdc-options field from. | |

#### Returns

Returns non-0 iff renewable-ok flag is set in KDC-REQ.

### shishi_kdcreq_enc_tkt_in_skey_p ()

```
int
shishi_kdcreq_enc_tkt_in_skey_p (Shishi *handle,
                                 Shishi_asn1 kdcreq);
```

Determine if KDC-Option enc-tkt-in-skey flag is set.

This option is used only by the ticket-granting service. The ENC-TKT-IN-SKEY option indicates that the ticket for the end server is to be encrypted in the session key from the additional ticket-granting ticket provided.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to get kdc-options field from. |

#### Returns

Returns non-0 iff enc-tkt-in-skey flag is set in KDC-REQ.

### shishi_kdcreq_renew_p ()

```
int
shishi_kdcreq_renew_p (Shishi *handle,
                       Shishi_asn1 kdcreq);
```

Determine if KDC-Option renew flag is set.

This option is used only by the ticket-granting service. The RENEW option indicates that the present request is for a renewal. The ticket provided is encrypted in the secret key for the server on which it is valid. This option will only be honored if the ticket to be renewed has its RENEWABLE flag set and if the time in its renew-till field has not passed. The ticket to be renewed is passed in the padata field as part of the authentication header.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ variable to get kdc-options field from. |

#### Returns

Returns non-0 iff renew flag is set in KDC-REQ.

### shishi_kdcreq_validate_p ()

```
int
shishi_kdcreq_validate_p (Shishi *handle,
                          Shishi_asn1 kdcreq);
```

Determine if KDC-Option validate flag is set.

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

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to get kdc-options field from. | |

#### Returns

Returns non-0 iff validate flag is set in KDC-REQ.

### shishi_kdcreq_options_set ()

```
int
shishi_kdcreq_options_set (Shishi *handle,
                           Shishi_asn1 kdcreq,
                           uint32_t options);
```

Set options in KDC-REQ. Note that this reset any already existing flags.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to set etype field in. | |
| options | integer with flags to store in KDC-REQ. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_options_add ()

```
int
shishi_kdcreq_options_add (Shishi *handle,
                           Shishi_asn1 kdcreq,
                           uint32_t option);
```

Add KDC-Option to KDC-REQ. This preserves all existing options.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ variable to set etype field in. | |
| option | integer with options to add in KDC-REQ. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_clear_padata ()

```
int
shishi_kdcreq_clear_padata (Shishi *handle,
                            Shishi_asn1 kdcreq);
```

Remove the padata field from KDC-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ to remove PA-DATA from. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_get_padata ()

```
int
shishi_kdcreq_get_padata (Shishi *handle,
                          Shishi_asn1 kdcreq,
                          Shishi_padata_type padatatype,
                          char **out,
                          size_t *outlen);
```

Get pre authentication data (PA-DATA) from KDC-REQ. Pre authentication data is used to pass various information to KDC, such as in case of a SHISHI_PA_TGS_REQ padatatype the AP-REQ that authenticates the user to get the ticket.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ to get PA-DATA from. |
| padatatype | type of PA-DATA, see Shishi_padata_type. |
| out | output array with newly allocated PA-DATA value. |
| outlen | size of output array with PA-DATA value. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_get_padata_tgs ()

```
int
shishi_kdcreq_get_padata_tgs (Shishi *handle,
                              Shishi_asn1 kdcreq,
                              Shishi_asn1 *apreq);
```

Extract TGS pre-authentication data from KDC-REQ. The data is an AP-REQ that authenticates the request. This function call shishi_kdcreq_get_padata() with a SHISHI_PA_TGS_REQ padatatype and DER decode the result (if any).

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ to get PA-TGS-REQ from. | |
| apreq | Output variable with newly allocated AP-REQ. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_add_padata ()

```
int
shishi_kdcreq_add_padata (Shishi *handle,
                          Shishi_asn1 kdcreq,
                          int padatatype,
                          const char *data,
                          size_t datalen);
```

Add new pre authentication data (PA-DATA) to KDC-REQ. This is used to pass various information to KDC, such as in case of a SHISHI_PA_TGS_REQ padatatype the AP-REQ that authenticates the user to get the ticket. (But also see shishi_kdcreq_add_padata_tgs which takes an AP-REQ directly.)

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ to add PA-DATA to. | |
| padatatype | type of PA-DATA, see Shishi_padata_type. | |
| data | input array with PA-DATA value. | |
| datalen | size of input array with PA-DATA value. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_add_padata_tgs ()

```
int
shishi_kdcreq_add_padata_tgs (Shishi *handle,
                              Shishi_asn1 kdcreq,
                              Shishi_asn1 apreq);
```

Add TGS pre-authentication data to KDC-REQ. The data is an AP-REQ that authenticates the request. This functions simply DER encodes the AP-REQ and calls shishi_kdcreq_add_padata() with a SHISHI_PA_TGS_REQ padatatype.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ to add PA-DATA to. | |
| apreq | AP-REQ to add as PA-DATA. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_add_padata_preauth ()

```
int
shishi_kdcreq_add_padata_preauth (Shishi *handle,
                                  Shishi_asn1 kdcreq,
                                  Shishi_key *key);
```

Add pre-authentication data to KDC-REQ.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcreq | KDC-REQ to add pre-authentication data to. | |
| key | Key used to encrypt pre-auth data. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcreq_build ()

```
int
shishi_kdcreq_build (Shishi *handle,
                     Shishi_asn1 kdcreq);
```

### shishi_as_derive_salt ()

```
int
shishi_as_derive_salt (Shishi *handle,
                       Shishi_asn1 asreq,
                       Shishi_asn1 asrep,
                       char **salt,
                       size_t *saltlen);
```

Computes the salt that should be used when deriving a key via shishi_string_to_key() for an AS exchange. Currently this searches for PA-DATA of type SHISHI_PA_PW_SALT in the AS-REP provided by `asrep` , and if present returns it. Otherwise the salt is composed from the client name and the realm, both are extracted from the request `asreq` .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |
| asreq | Input AS-REQ variable. | |
| asrep | Input AS-REP variable. | |
| salt | Returned pointer to newly allocated output array. | |
| saltlen | Pointer to integer, returning size of output array. | |

#### Returns

Returns SHISHI_OK if successful. Failure conditions include various ASN.1 issues.

### shishi_tgs_process ()

```
int
shishi_tgs_process (Shishi *handle,
                    Shishi_asn1 tgsreq,
                    Shishi_asn1 tgsrep,
                    Shishi_asn1 authenticator,
                    Shishi_asn1 oldenckdcreppart,
                    Shishi_asn1 *enckdcreppart);
```

Processes a TGS client exchange and outputs the decrypted EncKDCRepPart, holding details about the received ticket. This function simply derives the encryption key from the ticket used to construct the original TGS request, and then calls shishi_kdc_process().

#### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |
| tgsreq | Input variable holding the transmitted KDC-REQ. | |
| tgsrep | Input variable holding the received KDC-REP. | |
| authenticator | Input variable with an authenticator extracted from the AP-REQ part of `tgsreq` . | |

| | | |
|---|---|---|
| oldenckdcreppart | Input variable with EncKDCRepPart used in the request. | |
| enckdcreppart | Output variable holding the new EncKDCRepPart. | |

### Returns

Returns SHISHI_OK if the TGS client exchange was successful. Failures include ASN.1 and TGS conditions.

### shishi_as_process ()

```
int
shishi_as_process (Shishi *handle,
                   Shishi_asn1 asreq,
                   Shishi_asn1 asrep,
                   const char *string,
                   Shishi_asn1 *enckdcreppart);
```

Processes an AS client exchange and returns the decrypted EncKDCRepPart, holding details about the received ticket. This function simply derives the encryption key from the password, and then calls shishi_kdc_process().

### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |
| asreq | Input variable holding the transmitted KDC-REQ. | |
| asrep | Input variable holding the received KDC-REP. | |
| string | Input variable with a null terminated password. | |
| enckdcreppart | Output variable returning a new EncKDCRepPart. | |

### Returns

Returns SHISHI_OK if the AS client exchange was successful. Multiple failure conditions are possible.

### shishi_kdc_process ()

```
int
shishi_kdc_process (Shishi *handle,
                    Shishi_asn1 kdcreq,
                    Shishi_asn1 kdcrep,
                    Shishi_key *key,
                    int keyusage,
                    Shishi_asn1 *enckdcreppart);
```

Processes a KDC client exchange and extracts a decrypted EncKDCRepPart, holding details about the received ticket. Use shishi_kdcrep_get_ticket() to extract the ticket itself. This function verifies the various conditions that must hold if the response is to be considered valid. In particular, it compares nonces (using shishi_kdc_check_nonce()), and if the exchange was an AS exchange, it also checks cname and crealm (using shishi_as_check_cname(), shishi_as_check_crealm()).

Usually shishi_as_process() and shishi_tgs_process() should be used instead of this call, since they simplify computation of the decryption key.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |
| kdcreq | Input variable holding the transmitted KDC-REQ. | |
| kdcrep | Input variable holding the received KDC-REP. | |
| key | Input pointet to key for decrypting parts of *kdcrep* . | |
| keyusage | Kerberos key usage code. | |
| enckdcreppart | Output pointer for the extracted EncKDCRepPart. | |

#### Returns

Returns SHISHI_OK if the KDC client exchange was successful. Multiple failure conditions are possible.

### shishi_kdcreq_sendrecv ()

```
int
shishi_kdcreq_sendrecv (Shishi *handle,
                        Shishi_asn1 kdcreq,
                        Shishi_asn1 *kdcrep);
```

Sends a request to KDC, and receives the response. The provided AS-REQ, in *kdcreq* , sets all data for the request. On reception the reply is decoded as AS-REP into *kdcrep* .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| kdcreq | Input variable with a prepared AS-REQ. | |
| kdcrep | Output pointer variable returning received AS-REP. | |

#### Returns

Return code is SHISHI_OK on success, SHISHI_KDC_TIMEOUT on timeouts, SHISHI_ASN1_ERROR on translation errors, and SHISHI_GOT_KRBERROR for other corruptions.

### shishi_kdcreq_sendrecv_hint ()

```
int
shishi_kdcreq_sendrecv_hint (Shishi *handle,
                             Shishi_asn1 kdcreq,
                             Shishi_asn1 *kdcrep,
                             Shishi_tkts_hint *hint);
```

Sends a request to KDC, and receives the response. The provided request *kdcreq* and the hints structure *hint* , together determine transmitted data. On reception the reply is decoded as AS-REP into *kdcrep* .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle created by shishi_init(). | |
| kdcreq | Input variable with a prepared AS-REQ. | |
| kdcrep | Output pointer variable for decoded AS-REP. | |
| hint | Input Shishi_tkts_hint structure with flags. | |

#### Returns

Return code is SHISHI_OK on success, SHISHI_KDC_TIMEOUT on timeouts, SHISHI_ASN1_ERROR on translation errors, and SHISHI_GOT_KRBERROR for other corruptions.

### shishi_kdc_copy_crealm ()

```
int
shishi_kdc_copy_crealm (Shishi *handle,
                        Shishi_asn1 kdcrep,
                        Shishi_asn1 encticketpart);
```

Reads the field "crealm" from the ticket `encticketpart` and copies the value into the reply `kdcrep` .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |
| kdcrep | KDC-REP where the field "crealm" is updated. | |
| encticketpart | EncTicketPart providing "crealm" field. | |

#### Returns

Returns SHISHI_OK if successful, and ASN.1 failures otherwise.

### shishi_as_check_crealm ()

```
int
shishi_as_check_crealm (Shishi *handle,
                        Shishi_asn1 asreq,
                        Shishi_asn1 asrep);
```

Verifies that the fields `asreq.req` -body.realm and `asrep.crealm` contain identical realm names. This is one of the steps that has to be performed when processing an exchange of AS-REQ and AS-REP; see shishi_kdc_process() for more details.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |

| | |
|---|---|
| asreq | Request of type AS-REQ. |
| asrep | Reply structure of type AS-REP. |

### Returns

Returns SHISHI_OK if successful, SHISHI_REALM_MISMATCH whenever the realm names differ, and an error code otherwise.

### shishi_kdc_copy_cname ()

```
int
shishi_kdc_copy_cname (Shishi *handle,
                       Shishi_asn1 kdcrep,
                       Shishi_asn1 encticketpart);
```

Reads the field "cname" from the ticket `encticketpart` and copies the value into the reply `kdcrep` .

### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |
| kdcrep | KDC-REP where the field "cname" is updated. |
| encticketpart | EncTicketPart providing "cname" field. |

### Returns

Returns SHISHI_OK if successful, and ASN.1 failures otherwise.

### shishi_as_check_cname ()

```
int
shishi_as_check_cname (Shishi *handle,
                       Shishi_asn1 asreq,
                       Shishi_asn1 asrep);
```

Verifies that the fields `asreq.req` -body.cname and `asrep.cname` contain identical names. This is one of the steps that has to be performed when processing an exchange of AS-REQ and AS-REP; see shishi_kdc_process() for more details.

### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |
| asreq | Request of type AS-REQ. |
| asrep | Reply structure of type AS-REP. |

### Returns

Returns SHISHI_OK if successful, SHISHI_CNAME_MISMATCH if the names differ, and an error code otherwise.

### shishi_kdc_copy_nonce ()

```
int
shishi_kdc_copy_nonce (Shishi *handle,
                       Shishi_asn1 kdcreq,
                       Shishi_asn1 enckdcreppart);
```

Sets the field "nonce" in *enckdcreppart* to a value retreived from the corresponding field in *kdcreq* .

#### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |
| kdcreq | KDC-REQ providing "nonce" field. |
| enckdcreppart | EncKDCRepPart where "nonce" field is updated. |

#### Returns

Returns SHISHI_OK if successful.

### shishi_kdc_check_nonce ()

```
int
shishi_kdc_check_nonce (Shishi *handle,
                        Shishi_asn1 kdcreq,
                        Shishi_asn1 enckdcreppart);
```

Verifies that *kdcreq.req* -body.nonce and *enckdcreppart.nonce* contain matching values. This is one of the steps that has to be performed when processing an exchange of KDC-REQ and KDC-REP.

#### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |
| kdcreq | Request of type KDC-REQ. |
| enckdcreppart | Encrypted KDC-REP part. |

#### Returns

Returns SHISHI_OK if successful, SHISHI_NONCE_MISMATCH whenever the nonces are of differing lengths (usually a sign that a buggy server truncates the nonce to 4 bytes) and the same code if the nonce values differ, or an error code otherwise.

### shishi_asrep ()

```
Shishi_asn1
shishi_asrep (Shishi *handle);
```

This function creates a new AS-REP, populated with some default values.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the AS-REP or NULL on failure.

### shishi_tgsrep ()

```
Shishi_asn1
shishi_tgsrep (Shishi *handle);
```

This function creates a new TGS-REP, populated with some default values.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the TGS-REP or NULL on failure.

### shishi_kdcrep_save ()

```
int
shishi_kdcrep_save (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 kdcrep);
```

Print DER encoding of KDC-REP to file.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for writing. |
| kdcrep | KDC-REP to save. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_print ()

```
int
shishi_kdcrep_print (Shishi *handle,
                     FILE *fh,
                     Shishi_asn1 kdcrep);
```

Print ASCII armored DER encoding of KDC-REP to file.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for writing. |
| kdcrep | KDC-REP to print. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_to_file ()

```
int
shishi_kdcrep_to_file (Shishi *handle,
                       Shishi_asn1 kdcrep,
                       int filetype,
                       const char *filename);
```

Write KDC-REP to file in specified TYPE. The file will be truncated if it exists.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcrep | KDC-REP to save. |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. |
| filename | input variable with filename to write to. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_parse ()

```
int
shishi_kdcrep_parse (Shishi *handle,
                     FILE *fh,
                     Shishi_asn1 *kdcrep);
```

Read ASCII armored DER encoded KDC-REP from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| kdcrep | output variable with newly allocated KDC-REP. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_read ()

```
int
shishi_kdcrep_read (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 *kdcrep);
```

Read DER encoded KDC-REP from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| kdcrep | output variable with newly allocated KDC-REP. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_from_file ()

```
int
shishi_kdcrep_from_file (Shishi *handle,
                         Shishi_asn1 *kdcrep,
                         int filetype,
                         const char *filename);
```

Read KDC-REP from file in specified TYPE.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcrep | output variable with newly allocated KDC-REP. |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. |
| filename | input variable with filename to read from. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_clear_padata ()

```
int
shishi_kdcrep_clear_padata (Shishi *handle,
                            Shishi_asn1 kdcrep);
```

Remove the padata field from KDC-REP.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcrep | KDC-REP to remove PA-DATA from. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_get_enc_part_etype ()

```
int
shishi_kdcrep_get_enc_part_etype (Shishi *handle,
                                  Shishi_asn1 kdcrep,
                                  int32_t *etype);
```

Extract KDC-REP.enc-part.etype.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcrep | KDC-REP variable to get value from. | |
| etype | output variable that holds the value. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_add_enc_part ()

```
int
shishi_kdcrep_add_enc_part (Shishi *handle,
                            Shishi_asn1 kdcrep,
                            Shishi_key *key,
                            int keyusage,
                            Shishi_asn1 enckdcreppart);
```

Encrypts DER encoded EncKDCRepPart using key and stores it in the KDC-REP.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcrep | KDC-REP to add enc-part field to. |
| key | key used to encrypt enc-part. |
| keyusage | key usage to use, normally SHISHI_KEYUSAGE_ENCASREPPART, SHISHI_KEYUSAGE_ENCTGSREPPART_SESSION_KEY or SHISHI_KEYUSAGE_ENCTGSREPPART_AUTHENTICATOR_KEY. |
| enckdcreppart | EncKDCRepPart to add. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_get_ticket ()

```
int
shishi_kdcrep_get_ticket (Shishi *handle,
                          Shishi_asn1 kdcrep,
                          Shishi_asn1 *ticket);
```

Extract ticket from KDC-REP.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcrep | KDC-REP variable to get ticket from. |
| ticket | output variable to hold extracted ticket. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_set_ticket ()

```
int
shishi_kdcrep_set_ticket (Shishi *handle,
                          Shishi_asn1 kdcrep,
                          Shishi_asn1 ticket);
```

Copy ticket into KDC-REP.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

| | | |
|---|---|---|
| kdcrep | KDC-REP to add ticket field to. | |
| ticket | input ticket to copy into KDC-REP ticket field. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_crealm_set ()

```
int
shishi_kdcrep_crealm_set (Shishi *handle,
                          Shishi_asn1 kdcrep,
                          const char *crealm);
```

Set the client realm field in the KDC-REP.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcrep | Kdcrep variable to set realm field in. | |
| crealm | input array with name of realm. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_cname_set ()

```
int
shishi_kdcrep_cname_set (Shishi *handle,
                         Shishi_asn1 kdcrep,
                         Shishi_name_type name_type,
                         const char *cname[]);
```

Set the client name field in the KDC-REP.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcrep | Kdcrep variable to set server name field in. | |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. | |
| cname | input array with principal name. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_client_set ()

```
int
shishi_kdcrep_client_set (Shishi *handle,
                          Shishi_asn1 kdcrep,
                          const char *client);
```

Set the client name field in the KDC-REP.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| kdcrep | Kdcrep variable to set server name field in. | |
| client | zero-terminated string with principal name on RFC 1964 form. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_crealmserver_set ()

```
int
shishi_kdcrep_crealmserver_set (Shishi *handle,
                                Shishi_asn1 kdcrep,
                                const char *crealm,
                                const char *client);
```

### shishi_kdcrep_set_enc_part ()

```
int
shishi_kdcrep_set_enc_part (Shishi *handle,
                            Shishi_asn1 kdcrep,
                            int32_t etype,
                            uint32_t kvno,
                            const char *buf,
                            size_t buflen);
```

Set the encrypted enc-part field in the KDC-REP. The encrypted data is usually created by calling shishi_encrypt() on the DER encoded enc-part. To save time, you may want to use shishi_kdcrep_add_enc_part() instead, which calculates the encrypted data and calls this function in one step.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| kdcrep | KDC-REP to add enc-part field to. |
| etype | encryption type used to encrypt enc-part. |
| kvno | key version number. |
| buf | input array with encrypted enc-part. |
| buflen | size of input array with encrypted enc-part. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdcrep_decrypt ()

```
int
shishi_kdcrep_decrypt (Shishi *handle,
                       Shishi_asn1 kdcrep,
                       Shishi_key *key,
                       int keyusage,
                       Shishi_asn1 *enckdcreppart);
```

### shishi_enckdcreppart ()

```
Shishi_asn1
shishi_enckdcreppart (Shishi *handle);
```

### shishi_encasreppart ()

```
Shishi_asn1
shishi_encasreppart (Shishi *handle);
```

### shishi_enckdcreppart_get_key ()

```
int
shishi_enckdcreppart_get_key (Shishi *handle,
                              Shishi_asn1 enckdcreppart,
                              Shishi_key **key);
```

Extract the key to use with the ticket sent in the KDC-REP associated with the EncKDCRepPart input variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| enckdcreppart | input EncKDCRepPart variable. |
| key | newly allocated encryption key handle. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_key_set ()

```
int
shishi_enckdcreppart_key_set (Shishi *handle,
                              Shishi_asn1 enckdcreppart,
                              Shishi_key *key);
```

Set the EncKDCRepPart.key field to key type and value of supplied key.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| enckdcreppart | input EncKDCRepPart variable. | |
| key | key handle with information to store in enckdcreppart. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_nonce_set ()

```
int
shishi_enckdcreppart_nonce_set (Shishi *handle,
                                Shishi_asn1 enckdcreppart,
                                uint32_t nonce);
```

Set the EncKDCRepPart.nonce field.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| enckdcreppart | input EncKDCRepPart variable. | |
| nonce | nonce to set in EncKDCRepPart. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_flags_set ()

```
int
shishi_enckdcreppart_flags_set (Shishi *handle,
                                  Shishi_asn1 enckdcreppart,
                                  int flags);
```

Set the EncKDCRepPart.flags field.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| enckdcreppart | input EncKDCRepPart variable. | |
| flags | flags to set in EncKDCRepPart. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_authtime_set ()

```
int
shishi_enckdcreppart_authtime_set (Shishi *handle,
                                   Shishi_asn1 enckdcreppart,
                                   const char *authtime);
```

Set the EncTicketPart.authtime to supplied value.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| enckdcreppart | input EncKDCRepPart variable. | |
| authtime | character buffer containing a generalized time string. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_starttime_set ()

```
int
shishi_enckdcreppart_starttime_set (Shishi *handle,
                                    Shishi_asn1 enckdcreppart,
                                    const char *starttime);
```

Set the EncTicketPart.starttime to supplied value. Use a NULL value for `starttime` to remove the field.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| enckdcreppart | input EncKDCRepPart variable. |
| starttime | character buffer containing a generalized time string. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_endtime_set ()

```
int
shishi_enckdcreppart_endtime_set (Shishi *handle,
                                  Shishi_asn1 enckdcreppart,
                                  const char *endtime);
```

Set the EncTicketPart.endtime to supplied value.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| enckdcreppart | input EncKDCRepPart variable. |
| endtime | character buffer containing a generalized time string. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_renew_till_set ()

```
int
shishi_enckdcreppart_renew_till_set (Shishi *handle,
                                     Shishi_asn1 enckdcreppart,
                                     const char *renew_till);
```

Set the EncTicketPart.renew-till to supplied value. Use a NULL value for *renew_till* to remove the field.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| enckdcreppart | input EncKDCRepPart variable. |
| renew_till | character buffer containing a generalized time string. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_srealm_set ()

```
int
shishi_enckdcreppart_srealm_set (Shishi *handle,
                                 Shishi_asn1 enckdcreppart,
                                 const char *srealm);
```

Set the server realm field in the EncKDCRepPart.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| enckdcreppart | EncKDCRepPart variable to set realm field in. | |
| srealm | input array with name of realm. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_sname_set ()

```
int
shishi_enckdcreppart_sname_set (Shishi *handle,
                                Shishi_asn1 enckdcreppart,
                                Shishi_name_type name_type,
                                char *sname[]);
```

Set the server name field in the EncKDCRepPart.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| enckdcreppart | EncKDCRepPart variable to set server name field in. | |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. | |
| sname | input array with principal name. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_enckdcreppart_server_set ()

```
int
shishi_enckdcreppart_server_set (Shishi *handle,
                                 Shishi_asn1 enckdcreppart,
                                 const char *server);
```

### shishi_enckdcreppart_srealmserver_set ()

```
int
shishi_enckdcreppart_srealmserver_set (Shishi *handle,
                                       Shishi_asn1 enckdcreppart,
                                       const char *srealm,
                                       const char *server);
```

### shishi_enckdcreppart_populate_encticketpart ()

```
int
shishi_enckdcreppart_populate_encticketpart
                               (Shishi *handle,
                                Shishi_asn1 enckdcreppart,
                                Shishi_asn1 encticketpart);
```

Set the flags, authtime, starttime, endtime, renew-till and caddr fields of the EncKDCRepPart to the corresponding values in the EncTicketPart.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| enckdcreppart | input EncKDCRepPart variable. | |
| encticketpart | input EncTicketPart variable. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror ()

```
Shishi_asn1
shishi_krberror (Shishi *handle);
```

This function creates a new KRB-ERROR, populated with some default values.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |

#### Returns

Returns the KRB-ERROR or NULL on failure.

### shishi_krberror_print ()

```
int
shishi_krberror_print (Shishi *handle,
                       FILE *fh,
                       Shishi_asn1 krberror);
```

Print ASCII armored DER encoding of KRB-ERROR to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| krberror | KRB-ERROR to print. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_save ()

```
int
shishi_krberror_save (Shishi *handle,
                      FILE *fh,
                      Shishi_asn1 krberror);
```

Save DER encoding of KRB-ERROR to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| krberror | KRB-ERROR to save. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_to_file ()

```
int
shishi_krberror_to_file (Shishi *handle,
                         Shishi_asn1 krberror,
                         int filetype,
                         const char *filename);
```

Write KRB-ERROR to file in specified TYPE. The file will be truncated if it exists.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | KRB-ERROR to save. |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. |
| filename | input variable with filename to write to. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_parse ()

```
int
shishi_krberror_parse (Shishi *handle,
                       FILE *fh,
                       Shishi_asn1 *krberror);
```

Read ASCII armored DER encoded KRB-ERROR from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| krberror | output variable with newly allocated KRB-ERROR. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_read ()

```
int
shishi_krberror_read (Shishi *handle,
                      FILE *fh,
                      Shishi_asn1 *krberror);
```

Read DER encoded KRB-ERROR from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| krberror | output variable with newly allocated KRB-ERROR. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_from_file ()

```
int
shishi_krberror_from_file (Shishi *handle,
                           Shishi_asn1 *krberror,
                           int filetype,
                           const char *filename);
```

Read KRB-ERROR from file in specified TYPE.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | output variable with newly allocated KRB-ERROR. | |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. | |
| filename | input variable with filename to read from. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_build ()

```
int
shishi_krberror_build (Shishi *handle,
                       Shishi_asn1 krberror);
```

Finish KRB-ERROR, called before e.g. shishi_krberror_der. This function removes empty but OPTIONAL fields (such as cname), and

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_der ()

```
int
shishi_krberror_der (Shishi *handle,
                     Shishi_asn1 krberror,
                     char **out,
                     size_t *outlen);
```

DER encode KRB-ERROR. The caller must deallocate the OUT buffer.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| out | output array with newly allocated DER encoding of KRB-ERROR. | |
| outlen | length of output array with DER encoding of KRB-ERROR. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_crealm ()

```
int
shishi_krberror_crealm (Shishi *handle,
                        Shishi_asn1 krberror,
                        char **realm,
                        size_t *realmlen);
```

Extract client realm from KRB-ERROR.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| realm | output array with newly allocated name of realm in KRB-ERROR. | |
| realmlen | size of output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_remove_crealm ()

```
int
shishi_krberror_remove_crealm (Shishi *handle,
                               Shishi_asn1 krberror);
```

Remove client realm field in KRB-ERROR.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | krberror as allocated by shishi_krberror(). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_set_crealm ()

```
int
shishi_krberror_set_crealm (Shishi *handle,
                            Shishi_asn1 krberror,
                            const char *crealm);
```

Set realm field in krberror to specified value.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | krberror as allocated by shishi_krberror(). |
| crealm | input array with realm. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_client ()

```
int
shishi_krberror_client (Shishi *handle,
                        Shishi_asn1 krberror,
                        char **client,
                        size_t *clientlen);
```

Return client principal name in KRB-ERROR.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| client | pointer to newly allocated zero terminated string containing principal name. May be NULL (to only populate `clientlen` ). | |
| clientlen | pointer to length of `client` on output, excluding terminating zero. May be NULL (to only populate `client` ). | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_set_cname ()

```
int
shishi_krberror_set_cname (Shishi *handle,
                           Shishi_asn1 krberror,
                           Shishi_name_type name_type,
                           const char *cname[]);
```

Set principal field in krberror to specified value.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| name_type | type of principial, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. | |
| cname | input array with principal name. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_remove_cname ()

```
int
shishi_krberror_remove_cname (Shishi *handle,
                              Shishi_asn1 krberror);
```

Remove client realm field in KRB-ERROR.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_client_set ()

```
int
shishi_krberror_client_set (Shishi *handle,
                            Shishi_asn1 krberror,
                            const char *client);
```

Set the client name field in the Krberror.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | Krberror to set client name field in. | |
| client | zero-terminated string with principal name on RFC 1964 form. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_realm ()

```
int
shishi_krberror_realm (Shishi *handle,
                       Shishi_asn1 krberror,
                       char **realm,
                       size_t *realmlen);
```

Extract (server) realm from KRB-ERROR.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| realm | output array with newly allocated name of realm in KRB-ERROR. | |
| realmlen | size of output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_set_realm ()

```
int
shishi_krberror_set_realm (Shishi *handle,
                           Shishi_asn1 krberror,
                           const char *realm);
```

Set (server) realm field in krberror to specified value.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| realm | input array with (server) realm. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_server ()

```
int
shishi_krberror_server (Shishi *handle,
                        Shishi_asn1 krberror,
                        char **server,
                        size_t *serverlen);
```

Return server principal name in KRB-ERROR.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| server | pointer to newly allocated zero terminated string containing server name. May be NULL (to only populate `serverlen` ). | |
| serverlen | pointer to length of `server` on output, excluding terminating zero. May be NULL (to only populate `server` ). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_remove_sname ()

```
int
shishi_krberror_remove_sname (Shishi *handle,
                              Shishi_asn1 krberror);
```

Remove server name field in KRB-ERROR. (Since it is not marked OPTIONAL in the ASN.1 profile, what is done is to set the name-type to UNKNOWN and make sure the name-string sequence is empty.)

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | Krberror to set server name field in. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_set_sname ()

```
int
shishi_krberror_set_sname (Shishi *handle,
                           Shishi_asn1 krberror,
                           Shishi_name_type name_type,
                           const char *sname[]);
```

Set principal field in krberror to specified value.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. | |
| sname | input array with principal name. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_server_set ()

```
int
shishi_krberror_server_set (Shishi *handle,
                            Shishi_asn1 krberror,
                            const char *server);
```

Set the server name field in the Krberror.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | Krberror to set server name field in. | |
| server | zero-terminated string with principal name on RFC 1964 form. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_ctime ()

```
int
shishi_krberror_ctime (Shishi *handle,
                       Shishi_asn1 krberror,
                       char **t);
```

Extract client time from KRB-ERROR.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | Krberror to set client name field in. | |
| t | newly allocated zero-terminated output array with client time. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_ctime_set ()

```
int
shishi_krberror_ctime_set (Shishi *handle,
                           Shishi_asn1 krberror,
                           const char *t);
```

Store client time in Krberror.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | Krberror as allocated by shishi_krberror(). |
| t | string with generalized time value to store in Krberror. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_remove_ctime ()

```
int
shishi_krberror_remove_ctime (Shishi *handle,
                              Shishi_asn1 krberror);
```

Remove client time field in Krberror.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | Krberror as allocated by shishi_krberror(). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_cusec ()

```
int
shishi_krberror_cusec (Shishi *handle,
                       Shishi_asn1 krberror,
                       uint32_t *cusec);
```

Extract client microseconds field from Krberror.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | Krberror as allocated by shishi_krberror(). |
| cusec | output integer with client microseconds field. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_cusec_set ()

```
int
shishi_krberror_cusec_set (Shishi *handle,
                           Shishi_asn1 krberror,
                           uint32_t cusec);
```

Set the cusec field in the Krberror.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| cusec | client microseconds to set in krberror, 0-999999. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_remove_cusec ()

```
int
shishi_krberror_remove_cusec (Shishi *handle,
                              Shishi_asn1 krberror);
```

Remove client usec field in Krberror.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | Krberror as allocated by shishi_krberror(). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_stime ()

```
int
shishi_krberror_stime (Shishi *handle,
                       Shishi_asn1 krberror,
                       char **t);
```

Extract server time from KRB-ERROR.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | Krberror to set client name field in. | |
| t | newly allocated zero-terminated output array with server time. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_stime_set ()

```
int
shishi_krberror_stime_set (Shishi *handle,
                           Shishi_asn1 krberror,
                           const char *t);
```

Store server time in Krberror.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | Krberror as allocated by shishi_krberror(). | |
| t | string with generalized time value to store in Krberror. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_susec ()

```
int
shishi_krberror_susec (Shishi *handle,
                       Shishi_asn1 krberror,
                       uint32_t *susec);
```

Extract server microseconds field from Krberror.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | Krberror as allocated by shishi_krberror(). | |
| susec | output integer with server microseconds field. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_susec_set ()

```
int
shishi_krberror_susec_set (Shishi *handle,
                           Shishi_asn1 krberror,
                           uint32_t susec);
```

Set the susec field in the Krberror.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| susec | server microseconds to set in krberror, 0-999999. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_errorcode_set ()

```
int
shishi_krberror_errorcode_set (Shishi *handle,
                               Shishi_asn1 krberror,
                               int errorcode);
```

Set the error-code field to a new error code.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | KRB-ERROR structure with error code to set. | |
| errorcode | new error code to set in krberror. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_etext ()

```
int
shishi_krberror_etext (Shishi *handle,
                                   Shishi_asn1 krberror,
                                   char **etext,
                                   size_t *etextlen);
```

Extract additional error text from server (possibly empty).

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | KRB-ERROR structure with error code. | |
| etext | output array with newly allocated error text. | |
| etextlen | output length of error text. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_set_etext ()

```
int
shishi_krberror_set_etext (Shishi *handle,
                           Shishi_asn1 krberror,
                           const char *etext);
```

Set error text (e-text) field in KRB-ERROR to specified value.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| etext | input array with error text to set. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_remove_etext ()

```
int
shishi_krberror_remove_etext (Shishi *handle,
                              Shishi_asn1 krberror);
```

Remove error text (e-text) field in KRB-ERROR.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_edata ()

```
int
shishi_krberror_edata (Shishi *handle,
                       Shishi_asn1 krberror,
                       char **edata,
                       size_t *edatalen);
```

Extract additional error data from server (possibly empty).

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | KRB-ERROR structure with error code. | |
| edata | output array with newly allocated error data. | |
| edatalen | output length of error data. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_set_edata ()

```
int
shishi_krberror_set_edata (Shishi *handle,
                           Shishi_asn1 krberror,
                           const char *edata);
```

Set error text (e-data) field in KRB-ERROR to specified value.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | krberror as allocated by shishi_krberror(). | |
| edata | input array with error text to set. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_remove_edata ()

```
int
shishi_krberror_remove_edata (Shishi *handle,
                              Shishi_asn1 krberror);
```

Remove error text (e-data) field in KRB-ERROR.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | krberror as allocated by shishi_krberror(). |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_errorcode ()

```
int
shishi_krberror_errorcode (Shishi *handle,
                           Shishi_asn1 krberror,
                           int *errorcode);
```

Extract error code from KRB-ERROR.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | KRB-ERROR structure with error code. |
| errorcode | output integer KRB-ERROR error code. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_errorcode_fast ()

```
int
shishi_krberror_errorcode_fast (Shishi *handle,
                                Shishi_asn1 krberror);
```

Get error code from KRB-ERROR, without error checking.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| krberror | KRB-ERROR structure with error code. |

#### Returns

Return error code (see shishi_krberror_errorcode()) directly, or -1 on error.

### shishi_krberror_pretty_print ()

```
int
shishi_krberror_pretty_print (Shishi *handle,
                              FILE *fh,
                              Shishi_asn1 krberror);
```

Print KRB-ERROR error condition and some explanatory text to file descriptor.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle opened for writing. |
| krberror | KRB-ERROR structure with error code. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_krberror_errorcode_message ()

```
const char~*
shishi_krberror_errorcode_message (Shishi *handle,
                                   int errorcode);
```

Get human readable string describing KRB-ERROR code.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| errorcode | integer KRB-ERROR error code. |

#### Returns

Return a string describing error code. This function will always return a string even if the error code isn't known.

### shishi_krberror_message ()

```
const char~*
shishi_krberror_message (Shishi *handle,
                         Shishi_asn1 krberror);
```

Extract error code (see shishi_krberror_errorcode_fast()) and return error message (see shishi_krberror_errorcode_message()).

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | KRB-ERROR structure with error code. | |

#### Returns

Return a string describing error code. This function will always return a string even if the error code isn't known.

### shishi_krberror_methoddata ()

```
int
shishi_krberror_methoddata (Shishi *handle,
                            Shishi_asn1 krberror,
                            Shishi_asn1 *methoddata);
```

Extract METHOD-DATA ASN.1 object from the e-data field. The e-data field will only contain a METHOD-DATA if the krberror error code is SHISHI_KDC_ERR_PREAUTH_REQUIRED.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| krberror | KRB-ERROR structure with error code. | |
| methoddata | output ASN.1 METHOD-DATA. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_generalize_time ()

```
const char~*
shishi_generalize_time (Shishi *handle,
                        time_t t);
```

Converts C time `t` to a KerberosTime string representation. The returned string must not be deallocated by the caller.

#### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |
| t | C time to convert. |

### Returns

Returns a KerberosTime formatted string corresponding to the input parameter.

### shishi_generalize_now ()

```
const char~*
shishi_generalize_now (Shishi *handle);
```

Converts the current time to a KerberosTime string. The returned string must not be deallocated by the caller.

### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |

### Returns

Returns a KerberosTime formatted string corresponding to the current time.

### shishi_generalize_ctime ()

```
time_t
shishi_generalize_ctime (Shishi *handle,
                         const char *t);
```

Converts a KerberosTime formatted string in `t` to integral C time representation.

### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |
| t | KerberosTime string to convert. |

### Returns

Returns the C time corresponding to the input argument.

### shishi_time ()

```
int
shishi_time (Shishi *handle,
             Shishi_asn1 node,
             const char *field,
             char **t);
```

Extracts time information from an ASN.1 structure, and to be precise, does so from the named field *field* within the structure *node*.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |
| node | ASN.1 structure to get time from. | |
| field | Name of the field in the ASN.1 node carrying time. | |
| t | Returned pointer to an allocated char array containing a null-terminated time string. | |

#### Returns

Returns SHISHI_OK if successful, or an error.

### shishi_ctime ()

```
int
shishi_ctime (Shishi *handle,
              Shishi_asn1 node,
              const char *field,
              time_t *t);
```

Extracts time information from an ASN.1 structure `node` , and from an arbitrary element `field` of that structure.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |
| node | ASN.1 structure to read field from. | |
| field | Name of field in `node` to read. | |
| t | Pointer to a C-time valued integer, being updated with the time value to be extracted. | |

#### Returns

Returns SHISHI_OK if successful, SHISHI_ASN1_NO_ELEMENT if the element does not exist, SHISHI_ASN1_NO_VALUE if the field has no value. In all other cases, SHISHI_ASN1_ERROR is returned.

### shishi_randomize ()

```
int
shishi_randomize (Shishi *handle,
                  int strong,
                  void *data,
                  size_t datalen);
```

Store cryptographically random data of given size in the provided buffer.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| strong | 0 iff operation should not block, non-0 for very strong randomness. |
| data | output array to be filled with random data. |
| datalen | size of output array. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_crc ()

```
int
shishi_crc (Shishi *handle,
            const char *in,
            size_t inlen,
            char *out[4]);
```

Compute checksum of data using CRC32 modified according to RFC

1. The *out* buffer must be deallocated by the caller.

The modifications compared to standard CRC32 is that no initial and final XOR is performed, and that the output is returned in LSB-first order.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| in | input character array of data to checksum. |
| inlen | length of input character array of data to checksum. |
| out | newly allocated character array with checksum of data. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_md4 ()

```
int
shishi_md4 (Shishi *handle,
            const char *in,
            size_t inlen,
            char *out[16]);
```

Compute hash of data using MD4. The *out* buffer must be deallocated by the caller.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| in | input character array of data to hash. |
| inlen | length of input character array of data to hash. |
| out | newly allocated character array with hash of data. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_md5 ()

```
int
shishi_md5 (Shishi *handle,
            const char *in,
            size_t inlen,
            char *out[16]);
```

Compute hash of data using MD5. The *out* buffer must be deallocated by the caller.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| in | input character array of data to hash. |
| inlen | length of input character array of data to hash. |
| out | newly allocated character array with hash of data. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_hmac_md5 ()

```
int
shishi_hmac_md5 (Shishi *handle,
                 const char *key,
                 size_t keylen,
                 const char *in,
                 size_t inlen,
                 char *outhash[16]);
```

Compute keyed checksum of data using HMAC-MD5. The *outhash* buffer must be deallocated by the caller.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| key | input character array with key to use. |
| keylen | length of input character array with key to use. |
| in | input character array of data to hash. |
| inlen | length of input character array of data to hash. |
| outhash | newly allocated character array with keyed hash of data. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_hmac_sha1 ()

```
int
shishi_hmac_sha1 (Shishi *handle,
                  const char *key,
                  size_t keylen,
                  const char *in,
                  size_t inlen,
                  char *outhash[20]);
```

Compute keyed checksum of data using HMAC-SHA1. The *outhash* buffer must be deallocated by the caller.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| key | input character array with key to use. |
| keylen | length of input character array with key to use. |
| in | input character array of data to hash. |
| inlen | length of input character array of data to hash. |
| outhash | newly allocated character array with keyed hash of data. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_des_cbc_mac ()

```
int
shishi_des_cbc_mac (Shishi *handle,
                    const char key[8],
                    const char iv[8],
                    const char *in,
                    size_t inlen,
                    char *out[8]);
```

Computed keyed checksum of data using DES-CBC-MAC. The `out` buffer must be deallocated by the caller.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | input character array with key to use. | |
| iv | input character array with initialization vector to use, can be NULL. | |
| in | input character array of data to hash. | |
| inlen | length of input character array of data to hash. | |
| out | newly allocated character array with keyed hash of data. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_arcfour ()

```
int
shishi_arcfour (Shishi *handle,
                int decryptp,
                const char *key,
                size_t keylen,
                const char iv[258],
                char *ivout[258],
                const char *in,
                size_t inlen,
                char **out);
```

Encrypt or decrypt data (depending on `decryptp` ) using ARCFOUR. The `out` buffer must be deallocated by the caller.

The "initialization vector" used here is the concatenation of the sbox and i and j, and is thus always of size 256 + 1 + 1. This is a slight abuse of terminology, and assumes you know what you are doing. Don’t use it if you can avoid to.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |

| | | |
|---|---|---|
| decryptp | 0 to indicate encryption, non-0 to indicate decryption. | |
| key | input character array with key to use. | |
| keylen | length of input key array. | |
| iv | input character array with initialization vector to use, or NULL. | |
| ivout | output character array with updated initialization vector, or NULL. | |
| in | input character array of data to encrypt/decrypt. | |
| inlen | length of input character array of data to encrypt/decrypt. | |
| out | newly allocated character array with encrypted/decrypted data. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_des ()

```
int
shishi_des (Shishi *handle,
            int decryptp,
            const char key[8],
            const char iv[8],
            char *ivout[8],
            const char *in,
            size_t inlen,
            char **out);
```

Encrypt or decrypt data (depending on `decryptp` ) using DES in CBC mode. The `out` buffer must be deallocated by the caller.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| decryptp | 0 to indicate encryption, non-0 to indicate decryption. | |
| key | input character array with key to use. | |
| iv | input character array with initialization vector to use, or NULL. | |
| ivout | output character array with updated initialization vector, or NULL. | |
| in | input character array of data to encrypt/decrypt. | |

| | | |
|---|---|---|
| inlen | length of input character array of data to encrypt/decrypt. | |
| out | newly allocated character array with encrypted/decrypted data. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_3des ()

```
int
shishi_3des (Shishi *handle,
             int decryptp,
             const char key[24],
             const char iv[8],
             char *ivout[8],
             const char *in,
             size_t inlen,
             char **out);
```

Encrypt or decrypt data (depending on `decryptp` ) using 3DES in CBC mode. The `out` buffer must be deallocated by the caller.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| decryptp | 0 to indicate encryption, non-0 to indicate decryption. | |
| key | input character array with key to use. | |
| iv | input character array with initialization vector to use, or NULL. | |
| ivout | output character array with updated initialization vector, or NULL. | |
| in | input character array of data to encrypt/decrypt. | |
| inlen | length of input character array of data to encrypt/decrypt. | |
| out | newly allocated character array with encrypted/decrypted data. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_aes_cts ()

```
int
shishi_aes_cts (Shishi *handle,
                int decryptp,
                const char *key,
                size_t keylen,
                const char iv[16],
                char *ivout[16],
                const char *in,
                size_t inlen,
                char **out);
```

Encrypt or decrypt data (depending on `decryptp` ) using AES in CBC-CTS mode. The length of the key, `keylen` , decide if AES 128 or AES 256 should be used. The `out` buffer must be deallocated by the caller.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| decryptp | 0 to indicate encryption, non-0 to indicate decryption. | |
| key | input character array with key to use. | |
| keylen | length of input character array with key to use. | |
| iv | input character array with initialization vector to use, or NULL. | |
| ivout | output character array with updated initialization vector, or NULL. | |
| in | input character array of data to encrypt/decrypt. | |
| inlen | length of input character array of data to encrypt/decrypt. | |
| out | newly allocated character array with encrypted/decrypted data. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_cipher_supported_p ()

```
int
shishi_cipher_supported_p (int type);
```

Find out if cipher is supported.

#### Parameters

| | |
|---|---|
| type | encryption type, see Shishi_etype. |

#### Returns

Return 0 iff cipher is unsupported.

### shishi_cipher_name ()

```
const char~*
shishi_cipher_name (int type);
```

Read humanly readable string for cipher.

#### Parameters

| | |
|---|---|
| type | encryption type, see Shishi_etype. |

#### Returns

Return name of encryption type, e.g. "des3-cbc-sha1-kd", as defined in the standards.

### shishi_cipher_blocksize ()

```
int
shishi_cipher_blocksize (int type);
```

Get block size for cipher.

#### Parameters

| | |
|---|---|
| type | encryption type, see Shishi_etype. |

#### Returns

Return block size for encryption type, as defined in the standards.

### shishi_cipher_confoundersize ()

```
int
shishi_cipher_confoundersize (int type);
```

Get length of confounder for cipher.

#### Parameters

| | |
|---|---|
| type | encryption type, see Shishi_etype. |

#### Returns

Returns the size of the confounder (random data) for encryption type, as defined in the standards, or (size_t)-1 on error (e.g., unsupported encryption type).

### shishi_cipher_keylen ()

```
size_t
shishi_cipher_keylen (int type);
```

Get key length for cipher.

#### Parameters

| | |
|---|---|
| type | encryption type, see Shishi_etype. |

#### Returns

Return length of key used for the encryption type, as defined in the standards.

### shishi_cipher_randomlen ()

```
size_t
shishi_cipher_randomlen (int type);
```

Get length of random data for cipher.

#### Parameters

| | |
|---|---|
| type | encryption type, see Shishi_etype. |

#### Returns

Return length of random used for the encryption type, as defined in the standards, or (size_t)-1 on error (e.g., unsupported encryption type).

### shishi_cipher_defaultcksumtype ()

```
int
shishi_cipher_defaultcksumtype (int32_t type);
```

Get the default checksum associated with cipher.

#### Parameters

| | |
|---|---|
| type | encryption type, see Shishi_etype. |

#### Returns

Return associated checksum mechanism for the encryption type, as defined in the standards.

### shishi_cipher_parse ()

```
int
shishi_cipher_parse (const char *cipher);
```

Get cipher number by parsing string.

#### Parameters

| | |
|---|---|
| cipher | name of encryption type, e.g. "des3-cbc-sha1-kd". |

#### Returns

Return encryption type corresponding to a string.

### shishi_checksum_supported_p ()

```
int
shishi_checksum_supported_p (int32_t type);
```

Find out whether checksum is supported.

#### Parameters

| | |
|---|---|
| type | checksum type, see Shishi_cksumtype. |

#### Returns

Return 0 iff checksum is unsupported.

### shishi_checksum_name ()

```
const char~*
shishi_checksum_name (int32_t type);
```

Get name of checksum.

#### Parameters

| | |
|---|---|
| type | checksum type, see Shishi_cksumtype. |

#### Returns

Return name of checksum type, e.g. "hmac-sha1-96-aes256", as defined in the standards.

### shishi_checksum_cksumlen ()

```
size_t
shishi_checksum_cksumlen (int32_t type);
```

Get length of checksum output.

#### Parameters

| | |
|---|---|
| type | checksum type, see Shishi_cksumtype. |

#### Returns

Return length of checksum used for the checksum type, as defined in the standards.

### shishi_checksum_parse ()

```
int
shishi_checksum_parse (const char *checksum);
```

Get checksum number by parsing a string.

#### Parameters

| | |
|---|---|
| checksum | name of checksum type, e.g. "hmac-sha1-96-aes256". |

#### Returns

Return checksum type, see Shishi_cksumtype, corresponding to a string.

### shishi_string_to_key ()

```
int
shishi_string_to_key (Shishi *handle,
                      int32_t keytype,
                      const char *password,
                      size_t passwordlen,
                      const char *salt,
                      size_t saltlen,
                      const char *parameter,
                      Shishi_key *outkey);
```

Derive key from a string (password) and salt (commonly concatenation of realm and principal) for specified key type, and set the type and value in the given key to the computed values. The parameter value is specific for each keytype, and can be set if the parameter information is not available.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| keytype | cryptographic encryption type, see Shishi_etype. | |
| password | input array with password. | |
| passwordlen | length of input array with password. | |
| salt | input array with salt. | |
| saltlen | length of input array with salt. | |
| parameter | input array with opaque encryption type specific information. | |
| outkey | allocated key handle that will contain new key. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_random_to_key ()

```
int
shishi_random_to_key (Shishi *handle,
                      int32_t keytype,
                      const char *rnd,
                      size_t rndlen,
                      Shishi_key *outkey);
```

Derive key from random data for specified key type, and set the type and value in the given key to the computed values.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| keytype | cryptographic encryption type, see Shishi_etype. | |
| rnd | input array with random data. | |
| rndlen | length of input array with random data. | |
| outkey | allocated key handle that will contain new key. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encrypt_ivupdate_etype ()

```
int
shishi_encrypt_ivupdate_etype (Shishi *handle,
                               Shishi_key *key,
                               int keyusage,
                               int32_t etype,
                               const char *iv,
                               size_t ivlen,
                               char **ivout,
                               size_t *ivoutlen,
                               const char *in,
                               size_t inlen,
                               char **out,
                               size_t *outlen);
```

Encrypts data as per encryption method using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. If IVOUT or IVOUTLEN is NULL, the updated IV is not saved anywhere.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to encrypt with. | |
| keyusage | integer specifying what this key is encrypting. | |
| etype | integer specifying what cipher to use. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |
| ivout | output array with newly allocated updated initialization vector. | |
| ivoutlen | size of output array with updated initialization vector. | |
| in | input array with data to encrypt. | |
| inlen | size of input array with data to encrypt. | |
| out | output array with newly allocated encrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encrypt_iv_etype ()

```
int
shishi_encrypt_iv_etype (Shishi *handle,
                            Shishi_key *key,
                            int keyusage,
                            int32_t etype,
                            const char *iv,
                            size_t ivlen,
                            const char *in,
                            size_t inlen,
                            char **out,
                            size_t *outlen);
```

Encrypts data as per encryption method using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. The next IV is lost, see shishi_encrypt_ivupdate_etype if you need it.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to encrypt with. | |
| keyusage | integer specifying what this key is encrypting. | |
| etype | integer specifying what cipher to use. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |
| in | input array with data to encrypt. | |
| inlen | size of input array with data to encrypt. | |
| out | output array with newly allocated encrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encrypt_etype ()

```
int
shishi_encrypt_etype (Shishi *handle,
                      Shishi_key *key,
                      int keyusage,
                      int32_t etype,
                      const char *in,
                      size_t inlen,
                      char **out,
                      size_t *outlen);
```

Encrypts data as per encryption method using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. The default IV is used, see shishi_encrypt_iv_etype if you need to alter it. The next IV is lost, see shishi_encrypt_ivupdate_etype if you need it.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to encrypt with. | |
| keyusage | integer specifying what this key is encrypting. | |
| etype | integer specifying what cipher to use. | |
| in | input array with data to encrypt. | |
| inlen | size of input array with data to encrypt. | |
| out | output array with newly allocated encrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encrypt_ivupdate ()

```
int
shishi_encrypt_ivupdate (Shishi *handle,
                         Shishi_key *key,
                         int keyusage,
                         const char *iv,
                         size_t ivlen,
                         char **ivout,
                         size_t *ivoutlen,
                         const char *in,
                         size_t inlen,
                         char **out,
                         size_t *outlen);
```

Encrypts data using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. If IVOUT or IVOUTLEN is NULL, the updated IV is not saved anywhere.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to encrypt with. | |
| keyusage | integer specifying what this key is encrypting. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |
| ivout | output array with newly allocated updated initialization vector. | |
| ivoutlen | size of output array with updated initialization vector. | |
| in | input array with data to encrypt. | |
| inlen | size of input array with data to encrypt. | |
| out | output array with newly allocated encrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encrypt_iv ()

```
int
shishi_encrypt_iv (Shishi *handle,
                   Shishi_key *key,
                   int keyusage,
                   const char *iv,
                   size_t ivlen,
                   const char *in,
                   size_t inlen,
                   char **out,
                   size_t *outlen);
```

Encrypts data using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. The next IV is lost, see shishi_encrypt_ivupdate if you need it.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to encrypt with. | |

| | | |
|---|---|---|
| keyusage | integer specifying what this key is encrypting. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |
| in | input array with data to encrypt. | |
| inlen | size of input array with data to encrypt. | |
| out | output array with newly allocated encrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_encrypt ()

```
int
shishi_encrypt (Shishi *handle,
                Shishi_key *key,
                int keyusage,
                char *in,
                size_t inlen,
                char **out,
                size_t *outlen);
```

Encrypts data using specified key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. The default IV is used, see shishi_encrypt_iv if you need to alter it. The next IV is lost, see shishi_encrypt_ivupdate if you need it.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to encrypt with. | |
| keyusage | integer specifying what this key is encrypting. | |
| in | input array with data to encrypt. | |
| inlen | size of input array with data to encrypt. | |
| out | output array with newly allocated encrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_decrypt_ivupdate_etype ()

```
int
shishi_decrypt_ivupdate_etype (Shishi *handle,
                               Shishi_key *key,
                               int keyusage,
                               int32_t etype,
                               const char *iv,
                               size_t ivlen,
                               char **ivout,
                               size_t *ivoutlen,
                               const char *in,
                               size_t inlen,
                               char **out,
                               size_t *outlen);
```

Decrypts data as per encryption method using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. If IVOUT or IVOUTLEN is NULL, the updated IV is not saved anywhere.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to decrypt with. | |
| keyusage | integer specifying what this key is decrypting. | |
| etype | integer specifying what cipher to use. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |
| ivout | output array with newly allocated updated initialization vector. | |
| ivoutlen | size of output array with updated initialization vector. | |
| in | input array with data to decrypt. | |
| inlen | size of input array with data to decrypt. | |
| out | output array with newly allocated decrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_decrypt_iv_etype ()

```
int
shishi_decrypt_iv_etype (Shishi *handle,
                         Shishi_key *key,
                         int keyusage,
                         int32_t etype,
                         const char *iv,
                         size_t ivlen,
                         const char *in,
                         size_t inlen,
                         char **out,
                         size_t *outlen);
```

Decrypts data as per encryption method using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. The next IV is lost, see shishi_decrypt_ivupdate_etype if you need it.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to decrypt with. | |
| keyusage | integer specifying what this key is decrypting. | |
| etype | integer specifying what cipher to use. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |
| in | input array with data to decrypt. | |
| inlen | size of input array with data to decrypt. | |
| out | output array with newly allocated decrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_decrypt_etype ()

```
int
shishi_decrypt_etype (Shishi *handle,
                      Shishi_key *key,
                      int keyusage,
                      int32_t etype,
                      const char *in,
                      size_t inlen,
                      char **out,
                      size_t *outlen);
```

Decrypts data as per encryption method using specified key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. The default IV is used, see shishi_decrypt_iv_etype if you need to alter it. The next IV is lost, see shishi_decrypt_ivupdate_etype if you need it.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to decrypt with. | |
| keyusage | integer specifying what this key is decrypting. | |
| etype | integer specifying what cipher to use. | |
| in | input array with data to decrypt. | |
| inlen | size of input array with data to decrypt. | |
| out | output array with newly allocated decrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_decrypt_ivupdate ()

```
int
shishi_decrypt_ivupdate (Shishi *handle,
                         Shishi_key *key,
                         int keyusage,
                         const char *iv,
                         size_t ivlen,
                         char **ivout,
                         size_t *ivoutlen,
                         const char *in,
                         size_t inlen,
                         char **out,
                         size_t *outlen);
```

Decrypts data using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. If IVOUT or IVOUTLEN is NULL, the updated IV is not saved anywhere.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to decrypt with. | |
| keyusage | integer specifying what this key is decrypting. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |
| ivout | output array with newly allocated updated initialization vector. | |
| ivoutlen | size of output array with updated initialization vector. | |
| in | input array with data to decrypt. | |
| inlen | size of input array with data to decrypt. | |
| out | output array with newly allocated decrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_decrypt_iv ()

```
int
shishi_decrypt_iv (Shishi *handle,
                   Shishi_key *key,
                   int keyusage,
                   const char *iv,
                   size_t ivlen,
                   const char *in,
                   size_t inlen,
                   char **out,
                   size_t *outlen);
```

Decrypts data using specified initialization vector and key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. The next IV is lost, see shishi_decrypt_ivupdate_etype if you need it.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to decrypt with. | |
| keyusage | integer specifying what this key is decrypting. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |
| in | input array with data to decrypt. | |
| inlen | size of input array with data to decrypt. | |
| out | output array with newly allocated decrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_decrypt ()

```
int
shishi_decrypt (Shishi *handle,
                Shishi_key *key,
                int keyusage,
                const char *in,
                size_t inlen,
                char **out,
                size_t *outlen);
```

Decrypts data specified key. The key actually used is derived using the key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller. The default IV is used, see shishi_decrypt_iv if you need to alter it. The next IV is lost, see shishi_decrypt_ivupdate if you need it.

Note that DECRYPT(ENCRYPT(data)) does not necessarily yield data exactly. Some encryption types add pad to make the data fit into the block size of the encryption algorithm. Furthermore, the pad is not guaranteed to look in any special way, although existing implementations often pad with the zero byte. This means that you may have to "frame" data, so it is possible to infer the original length after decryption. Compare ASN.1 DER which contains such information.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to decrypt with. | |

| | | |
|---|---|---|
| keyusage | integer specifying what this key is decrypting. | |
| in | input array with data to decrypt. | |
| inlen | size of input array with data to decrypt. | |
| out | output array with newly allocated decrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_checksum ()

```
int
shishi_checksum (Shishi *handle,
                 Shishi_key *key,
                 int keyusage,
                 int32_t cksumtype,
                 const char *in,
                 size_t inlen,
                 char **out,
                 size_t *outlen);
```

Integrity protect data using key, possibly altered by supplied key usage. If key usage is 0, no key derivation is used. The OUT buffer must be deallocated by the caller.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to compute checksum with. | |
| keyusage | integer specifying what this key is used for. | |
| cksumtype | the checksum algorithm to use. | |
| in | input array with data to integrity protect. | |
| inlen | size of input array with data to integrity protect. | |
| out | output array with newly allocated integrity protected data. | |
| outlen | output variable with length of output array with checksum. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_verify ()

```
int
shishi_verify (Shishi *handle,
               Shishi_key *key,
               int keyusage,
               int cksumtype,
               const char *in,
               size_t inlen,
               const char *cksum,
               size_t cksumlen);
```

Verify checksum of data using key, possibly altered by supplied key usage. If key usage is 0, no key derivation is used.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to verify checksum with. | |
| keyusage | integer specifying what this key is used for. | |
| cksumtype | the checksum algorithm to use. | |
| in | input array with data that was integrity protected. | |
| inlen | size of input array with data that was integrity protected. | |
| cksum | input array with alleged checksum of data. | |
| cksumlen | size of input array with alleged checksum of data. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_dk ()

```
int
shishi_dk (Shishi *handle,
           Shishi_key *key,
           const char *prfconstant,
           size_t prfconstantlen,
           Shishi_key *derivedkey);
```

Derive a key from a key and a constant thusly: DK(KEY, PRFCONSTANT) = SHISHI_RANDOM-TO-KEY(SHISHI_DR(KEY, PRFCONSTANT)).

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | input cryptographic key to use. | |

| | | |
|---|---|---|
| prfconstant | input array with the constant string. | |
| prfconstantlen | size of input array with the constant string. | |
| derivedkey | pointer to derived key (allocated by caller). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_dr ()

```
int
shishi_dr (Shishi *handle,
           Shishi_key *key,
           const char *prfconstant,
           size_t prfconstantlen,
           char *derivedrandom,
           size_t derivedrandomlen);
```

Derive "random" data from a key and a constant thusly: DR(KEY, PRFCONSTANT) = TRUNCATE(DERIVEDRANDOMLEN, SHISHI_ENCRYPT(KEY, PRFCONSTANT)).

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | input array with cryptographic key to use. | |
| prfconstant | input array with the constant string. | |
| prfconstantlen | size of input array with the constant string. | |
| derivedrandom | output array with derived random data. | |
| derivedrandomlen | size of output array with derived random data. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_n_fold ()

```
int
shishi_n_fold (Shishi *handle,
               const char *in,
               size_t inlen,
               char *out,
               size_t outlen);
```

Fold data into a fixed length output array, with the intent to give each input bit approximately equal weight in determining the value of each output bit.

The algorithm is from "A Better Key Schedule For DES-like Ciphers" by Uri Blumenthal and Steven M. Bellovin, http://www.research.att
although the sample vectors provided by the paper are incorrect.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| in | input array with data to decrypt. | |
| inlen | size of input array with data to decrypt ("M"). | |
| out | output array with decrypted data. | |
| outlen | size of output array ("N"). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_pbkdf2_sha1 ()

```
int
shishi_pbkdf2_sha1 (Shishi *handle,
                    const char *P,
                    size_t Plen,
                    const char *S,
                    size_t Slen,
                    unsigned int c,
                    unsigned int dkLen,
                    char *DK);
```

Derive key using the PBKDF2 defined in PKCS5. PBKDF2 applies a pseudorandom function to derive keys. The length of the derived key is essentially unbounded. (However, the maximum effective search space for the derived key may be limited by the structure of the underlying pseudorandom function, which is this function is always SHA1.)

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| P | input password, an octet string | |
| Plen | length of password, an octet string | |
| S | input salt, an octet string | |
| Slen | length of salt, an octet string | |
| c | iteration count, a positive integer | |
| dkLen | intended length in octets of the derived key, a positive integer, at most (2^32 - 1) * hLen. The DK array must have room for this many characters. | |
| DK | output derived key, a dkLen-octet string | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_crypto ()

```
Shishi_crypto~*
shishi_crypto (Shishi *handle,
               Shishi_key *key,
               int keyusage,
               int32_t etype,
               const char *iv,
               size_t ivlen);
```

Initialize a crypto context. This store a key, keyusage, encryption type and initialization vector in a "context", and the caller can then use this context to perform encryption via shishi_crypto_encrypt() and decryption via shishi_crypto_encrypt() without supplying all those details again. The functions also takes care of propagating the IV between calls.

When the application no longer need to use the context, it should deallocate resources associated with it by calling shishi_crypto_close().

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| key | key to encrypt with. | |
| keyusage | integer specifying what this key will encrypt/decrypt. | |
| etype | integer specifying what cipher to use. | |
| iv | input array with initialization vector | |
| ivlen | size of input array with initialization vector. | |

#### Returns

Return a newly allocated crypto context.

### shishi_crypto_close ()

```
void
shishi_crypto_close (Shishi_crypto *ctx);
```

Deallocate resources associated with the crypto context.

#### Parameters

| | | |
|---|---|---|
| ctx | crypto context as returned by shishi_crypto(). | |

### shishi_crypto_encrypt ()

```
int
```

```
shishi_crypto_encrypt (Shishi_crypto *ctx,
                       const char *in,
                       size_t inlen,
                       char **out,
                       size_t *outlen);
```

Encrypt data, using information (e.g., key and initialization vector) from context. The IV is updated inside the context after this call.

When the application no longer need to use the context, it should deallocate resources associated with it by calling shishi_crypto_close().

#### Parameters

| | | |
|---|---|---|
| ctx | crypto context as returned by shishi_crypto(). | |
| in | input array with data to encrypt. | |
| inlen | size of input array with data to encrypt. | |
| out | output array with newly allocated encrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_crypto_decrypt ()

```
int
shishi_crypto_decrypt (Shishi_crypto *ctx,
                       const char *in,
                       size_t inlen,
                       char **out,
                       size_t *outlen);
```

Decrypt data, using information (e.g., key and initialization vector) from context. The IV is updated inside the context after this call.

When the application no longer need to use the context, it should deallocate resources associated with it by calling shishi_crypto_close().

#### Parameters

| | | |
|---|---|---|
| ctx | crypto context as returned by shishi_crypto(). | |
| in | input array with data to decrypt. | |
| inlen | size of input array with data to decrypt. | |
| out | output array with newly allocated decrypted data. | |
| outlen | output variable with size of newly allocated output array. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_check_version ()

```
const char~*
shishi_check_version (const char *req_version);
```

Checks that the installed library version is at least as recent as the one provided in *req_version* . The version string is formatted like "1.0.2".

Whenever NULL is passed to this function, the check is suppressed, but the library version is still returned.

#### Parameters

| | |
|---|---|
| req_version | Oldest acceptable version, or NULL. |

#### Returns

Returns the active library version, or NULL, should the running library be too old.

### shishi_prompt_password_func ()

```
int
(*shishi_prompt_password_func) (Shishi *handle,
                                char **s,
                                const char *format,
                                va_list ap);
```

### shishi_prompt_password_callback_set ()

```
void
shishi_prompt_password_callback_set (Shishi *handle,
                                     shishi_prompt_password_func cb);
```

Set a callback function that will be used by shishi_prompt_password() to query the user for a password. The function pointer can be retrieved using shishi_prompt_password_callback_get().

The *cb* function should follow the shishi_prompt_password_func prototype:

int prompt_password (Shishi * *handle* , char ***s* , const char **format* , va_list *ap* );

If the function returns 0, the *s* variable should contain a newly allocated string with the password read from the user.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| cb | function pointer to application password callback, a shishi_prompt_password_func type. |

### shishi_prompt_password_callback_get ()

```
shishi_prompt_password_func
shishi_prompt_password_callback_get (Shishi *handle);
```

Get the application password prompt function callback as set by shishi_prompt_password_callback_set().

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the callback, a shishi_prompt_password_func type, or NULL.

### shishi_prompt_password ()

```
int
shishi_prompt_password (Shishi *handle,
                        char **s,
                        const char *format,
                        ...);
```

Format and print a prompt, and read a password from user. The password is possibly converted (e.g., converted from Latin-1 to UTF-8, or processed using Stringprep profile) following any "stringprocess" keywords in configuration files.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| s | pointer to newly allocated output string with read password. | |
| format | printf(3) style format string. | |
| ... | printf(3) style arguments. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_asn1_number_of_elements ()

```
int
shishi_asn1_number_of_elements (Shishi *handle,
                                Shishi_asn1 node,
                                const char *field,
                                size_t *n);
```

### shishi_asn1_empty_p ()

```
int
shishi_asn1_empty_p (Shishi *handle,
                     Shishi_asn1 node,
                     const char *field);
```

### shishi_asn1_read ()

```
int
shishi_asn1_read (Shishi *handle,
                  Shishi_asn1 node,
                  const char *field,
                  char **data,
                  size_t *datalen);
```

Extract data stored in a ASN.1 field into a newly allocated buffer. The buffer will always be zero terminated, even though `datalen` will not include the added zero.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| node | ASN.1 variable to read field from. | |
| field | name of field in *node* to read. | |
| data | newly allocated output buffer that will hold ASN.1 field data. | |
| datalen | actual size of output buffer. | |

#### Returns

Returns SHISHI_OK if successful, SHISHI_ASN1_NO_ELEMENT if the element do not exist, SHISHI_ASN1_NO_VALUE if the field has no value, ot SHISHI_ASN1_ERROR otherwise.

### shishi_asn1_read_inline ()

```
int
shishi_asn1_read_inline (Shishi *handle,
                         Shishi_asn1 node,
                         const char *field,
                         char *data,
                         size_t *datalen);
```

Extract data stored in a ASN.1 field into a fixed size buffer allocated by caller.

Note that since it is difficult to predict the length of the field, it is often better to use shishi_asn1_read() instead.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |

| | | |
|---|---|---|
| node | ASN.1 variable to read field from. | |
| field | name of field in *node* to read. | |
| data | pre-allocated output buffer that will hold ASN.1 field data. | |
| datalen | on input, maximum size of output buffer, on output, actual size of output buffer. | |

#### Returns

Returns SHISHI_OK if successful, SHISHI_ASN1_NO_ELEMENT if the element do not exist, SHISHI_ASN1_NO_VALUE if the field has no value, ot SHISHI_ASN1_ERROR otherwise.

### shishi_asn1_read_integer ()

```
int
shishi_asn1_read_integer (Shishi *handle,
                          Shishi_asn1 node,
                          const char *field,
                          int *i);
```

### shishi_asn1_read_int32 ()

```
int
shishi_asn1_read_int32 (Shishi *handle,
                        Shishi_asn1 node,
                        const char *field,
                        int32_t *i);
```

### shishi_asn1_read_uint32 ()

```
int
shishi_asn1_read_uint32 (Shishi *handle,
                         Shishi_asn1 node,
                         const char *field,
                         uint32_t *i);
```

### shishi_asn1_read_bitstring ()

```
int
shishi_asn1_read_bitstring (Shishi *handle,
                            Shishi_asn1 node,
                            const char *field,
                            uint32_t *flags);
```

### shishi_asn1_read_optional ()

```
int
shishi_asn1_read_optional (Shishi *handle,
                           Shishi_asn1 node,
                           const char *field,
                           char **data,
                           size_t *datalen);
```

Extract data stored in a ASN.1 field into a newly allocated buffer. If the field does not exist (i.e., SHISHI_ASN1_NO_ELEMENT), this function set datalen to 0 and succeeds. Can be useful to read ASN.1 fields which are marked OPTIONAL in the grammar, if you want to avoid special error handling in your code.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| node | ASN.1 variable to read field from. | |
| field | name of field in *node* to read. | |
| data | newly allocated output buffer that will hold ASN.1 field data. | |
| datalen | actual size of output buffer. | |

#### Returns

Returns SHISHI_OK if successful, SHISHI_ASN1_NO_VALUE if the field has no value, ot SHISHI_ASN1_ERROR otherwise.

### shishi_asn1_write ()

```
int
shishi_asn1_write (Shishi *handle,
                   Shishi_asn1 node,
                   const char *field,
                   const char *data,
                   size_t datalen);
```

### shishi_asn1_write_integer ()

```
int
shishi_asn1_write_integer (Shishi *handle,
                           Shishi_asn1 node,
                           const char *field,
                           int n);
```

### shishi_asn1_write_int32 ()

```
int
shishi_asn1_write_int32 (Shishi *handle,
                         Shishi_asn1 node,
                         const char *field,
                         int32_t n);
```

### shishi_asn1_write_uint32 ()

```
int
shishi_asn1_write_uint32 (Shishi *handle,
                          Shishi_asn1 node,
                          const char *field,
                          uint32_t n);
```

### shishi_asn1_write_bitstring ()

```
int
shishi_asn1_write_bitstring (Shishi *handle,
                             Shishi_asn1 node,
                             const char *field,
                             uint32_t flags);
```

### shishi_asn1_done ()

```
void
shishi_asn1_done (Shishi *handle,
                  Shishi_asn1 node);
```

Deallocate resources associated with ASN.1 structure. Note that the node must not be used after this call.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| node | ASN.1 node to deallocate. |

### shishi_asn1_pa_enc_ts_enc ()

```
Shishi_asn1
shishi_asn1_pa_enc_ts_enc (Shishi *handle);
```

Create new ASN.1 structure for PA-ENC-TS-ENC.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_encrypteddata ()

```
Shishi_asn1
shishi_asn1_encrypteddata (Shishi *handle);
```

Create new ASN.1 structure for EncryptedData

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_padata ()

```
Shishi_asn1
shishi_asn1_padata (Shishi *handle);
```

Create new ASN.1 structure for PA-DATA.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_methoddata ()

```
Shishi_asn1
shishi_asn1_methoddata (Shishi *handle);
```

Create new ASN.1 structure for METHOD-DATA.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_etype_info ()

```
Shishi_asn1
shishi_asn1_etype_info (Shishi *handle);
```

Create new ASN.1 structure for ETYPE-INFO.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_etype_info2 ()

```
Shishi_asn1
shishi_asn1_etype_info2 (Shishi *handle);
```

Create new ASN.1 structure for ETYPE-INFO2.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_asreq ()

```
Shishi_asn1
shishi_asn1_asreq (Shishi *handle);
```

Create new ASN.1 structure for AS-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_asrep ()

```
Shishi_asn1
shishi_asn1_asrep (Shishi *handle);
```

Create new ASN.1 structure for AS-REP.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_tgsreq ()

```
Shishi_asn1
shishi_asn1_tgsreq (Shishi *handle);
```

Create new ASN.1 structure for TGS-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_tgsrep ()

```
Shishi_asn1
shishi_asn1_tgsrep (Shishi *handle);
```

Create new ASN.1 structure for TGS-REP.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_apreq ()

```
Shishi_asn1
shishi_asn1_apreq (Shishi *handle);
```

Create new ASN.1 structure for AP-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_aprep ()

```
Shishi_asn1
shishi_asn1_aprep (Shishi *handle);
```

Create new ASN.1 structure for AP-REP.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_ticket ()

```
Shishi_asn1
shishi_asn1_ticket (Shishi *handle);
```

Create new ASN.1 structure for Ticket.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_encapreppart ()

```
Shishi_asn1
shishi_asn1_encapreppart (Shishi *handle);
```

Create new ASN.1 structure for AP-REP.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_encticketpart ()

```
Shishi_asn1
shishi_asn1_encticketpart (Shishi *handle);
```

Create new ASN.1 structure for EncTicketPart.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_authenticator ()

```
Shishi_asn1
shishi_asn1_authenticator (Shishi *handle);
```

Create new ASN.1 structure for Authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_enckdcreppart ()

```
Shishi_asn1
shishi_asn1_enckdcreppart (Shishi *handle);
```

Create new ASN.1 structure for EncKDCRepPart.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_encasreppart ()

```
Shishi_asn1
shishi_asn1_encasreppart (Shishi *handle);
```

Create new ASN.1 structure for EncASRepPart.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_krberror ()

```
Shishi_asn1
shishi_asn1_krberror (Shishi *handle);
```

Create new ASN.1 structure for KRB-ERROR.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_krbsafe ()

```
Shishi_asn1
shishi_asn1_krbsafe (Shishi *handle);
```

Create new ASN.1 structure for KRB-SAFE.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_priv ()

```
Shishi_asn1
shishi_asn1_priv (Shishi *handle);
```

Create new ASN.1 structure for KRB-PRIV.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_encprivpart ()

```
Shishi_asn1
shishi_asn1_encprivpart (Shishi *handle);
```

Create new ASN.1 structure for EncKrbPrivPart.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns ASN.1 structure.

### shishi_asn1_to_der ()

```
int
shishi_asn1_to_der (Shishi *handle,
                    Shishi_asn1 node,
                    char **der,
                    size_t *len);
```

Extract newly allocated DER representation of specified ASN.1 data.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| node | ASN.1 data to convert to DER. |
| der | output array that holds DER encoding of `node` . |
| len | output variable with length of `der` output array. |

#### Returns

Returns SHISHI_OK if successful, or SHISHI_ASN1_ERROR if DER encoding fails (common reasons for this is that the ASN.1 is missing required values).

### shishi_asn1_to_der_field ()

```
int
shishi_asn1_to_der_field (Shishi *handle,
                          Shishi_asn1 node,
                          const char *field,
                          char **der,
                          size_t *len);
```

Extract newly allocated DER representation of specified ASN.1 field.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| node | ASN.1 data that have field to extract. | |
| field | name of field in `node` to extract. | |
| der | output array that holds DER encoding of `field` in `node` . | |
| len | output variable with length of `der` output array. | |

#### Returns

Returns SHISHI_OK if successful, or SHISHI_ASN1_ERROR if DER encoding fails (common reasons for this is that the ASN.1 is missing required values).

### shishi_asn1_msgtype ()

```
Shishi_msgtype
shishi_asn1_msgtype (Shishi *handle,
                     Shishi_asn1 node);
```

Determine msg-type of ASN.1 type of a packet. Currently this uses the msg-type field instead of the APPLICATION tag, but this may be changed in the future.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| node | ASN.1 type to get msg type for. | |

#### Returns

Returns msg-type of ASN.1 type, 0 on failure.

### shishi_der_msgtype ()

```
Shishi_msgtype
```

```
shishi_der_msgtype (Shishi *handle,
                    const char *der,
                    size_t derlen);
```

Determine msg-type of DER coded data of a packet.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns msg-type of DER data, 0 on failure.

### shishi_asn1_print ()

```
void
shishi_asn1_print (Shishi *handle,
                   Shishi_asn1 node,
                   FILE *fh);
```

Print ASN.1 structure in human readable form, typically for debugging purposes.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| node | ASN.1 data that have field to extract. | |
| fh | file descriptor to print to, e.g. stdout. | |

### shishi_der2asn1 ()

```
Shishi_asn1
shishi_der2asn1 (Shishi *handle,
                 const char *der,
                 size_t derlen);
```

Convert arbitrary DER data of a packet to a ASN.1 type.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |

| | |
|---|---|
| derlen | length of input character array with DER encoding. |

#### Returns

Returns newly allocate ASN.1 corresponding to DER data, or NULL on failure.

### shishi_der2asn1_padata ()

```
Shishi_asn1
shishi_der2asn1_padata (Shishi *handle,
                        const char *der,
                        size_t derlen);
```

Decode DER encoding of PA-DATA and create a ASN.1 structure.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_methoddata ()

```
Shishi_asn1
shishi_der2asn1_methoddata (Shishi *handle,
                            const char *der,
                            size_t derlen);
```

Decode DER encoding of METHOD-DATA and create a ASN.1 structure.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_etype_info ()

```
Shishi_asn1
shishi_der2asn1_etype_info (Shishi *handle,
                            const char *der,
                            size_t derlen);
```

Decode DER encoding of ETYPE-INFO and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_etype_info2 ()

```
Shishi_asn1
shishi_der2asn1_etype_info2 (Shishi *handle,
                             const char *der,
                             size_t derlen);
```

Decode DER encoding of ETYPE-INFO2 and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_ticket ()

```
Shishi_asn1
shishi_der2asn1_ticket (Shishi *handle,
                        const char *der,
                        size_t derlen);
```

Decode DER encoding of Ticket and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_encticketpart ()

```
Shishi_asn1
shishi_der2asn1_encticketpart (Shishi *handle,
                               const char *der,
                               size_t derlen);
```

Decode DER encoding of EncTicketPart and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_asreq ()

```
Shishi_asn1
shishi_der2asn1_asreq (Shishi *handle,
                       const char *der,
                       size_t derlen);
```

Decode DER encoding of AS-REQ and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_tgsreq ()

```
Shishi_asn1
shishi_der2asn1_tgsreq (Shishi *handle,
                        const char *der,
                        size_t derlen);
```

Decode DER encoding of TGS-REQ and create a ASN.1 structure.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_asrep ()

```
Shishi_asn1
shishi_der2asn1_asrep (Shishi *handle,
                       const char *der,
                       size_t derlen);
```

Decode DER encoding of AS-REP and create a ASN.1 structure.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_tgsrep ()

```
Shishi_asn1
shishi_der2asn1_tgsrep (Shishi *handle,
                        const char *der,
                        size_t derlen);
```

Decode DER encoding of TGS-REP and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_kdcrep ()

```
Shishi_asn1
shishi_der2asn1_kdcrep (Shishi *handle,
                        const char *der,
                        size_t derlen);
```

Decode DER encoding of KDC-REP and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_kdcreq ()

```
Shishi_asn1
shishi_der2asn1_kdcreq (Shishi *handle,
                        const char *der,
                        size_t derlen);
```

Decode DER encoding of AS-REQ, TGS-REQ or KDC-REQ and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_apreq ()

```
Shishi_asn1
shishi_der2asn1_apreq (Shishi *handle,
                       const char *der,
                       size_t derlen);
```

Decode DER encoding of AP-REQ and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_aprep ()

```
Shishi_asn1
shishi_der2asn1_aprep (Shishi *handle,
                       const char *der,
                       size_t derlen);
```

Decode DER encoding of AP-REP and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_authenticator ()

```
Shishi_asn1
shishi_der2asn1_authenticator (Shishi *handle,
                               const char *der,
                               size_t derlen);
```

Decode DER encoding of Authenticator and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_krberror ()

```
Shishi_asn1
shishi_der2asn1_krberror (Shishi *handle,
                          const char *der,
                          size_t derlen);
```

Decode DER encoding of KRB-ERROR and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_krbsafe ()

```
Shishi_asn1
shishi_der2asn1_krbsafe (Shishi *handle,
                         const char *der,
                         size_t derlen);
```

Decode DER encoding of KRB-SAFE and create a ASN.1 structure.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_priv ()

```
Shishi_asn1
shishi_der2asn1_priv (Shishi *handle,
                      const char *der,
                      size_t derlen);
```

Decode DER encoding of KRB-PRIV and create a ASN.1 structure.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_encasreppart ()

```
Shishi_asn1
shishi_der2asn1_encasreppart (Shishi *handle,
                              const char *der,
                              size_t derlen);
```

Decode DER encoding of EncASRepPart and create a ASN.1 structure.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_enctgsreppart ()

```
Shishi_asn1
shishi_der2asn1_enctgsreppart (Shishi *handle,
                               const char *der,
                               size_t derlen);
```

Decode DER encoding of EncTGSRepPart and create a ASN.1 structure.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_enckdcreppart ()

```
Shishi_asn1
shishi_der2asn1_enckdcreppart (Shishi *handle,
                               const char *der,
                               size_t derlen);
```

Decode DER encoding of EncKDCRepPart and create a ASN.1 structure.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| der | input character array with DER encoding. |
| derlen | length of input character array with DER encoding. |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_encapreppart ()

```
Shishi_asn1
shishi_der2asn1_encapreppart (Shishi *handle,
                              const char *der,
                              size_t derlen);
```

Decode DER encoding of EncAPRepPart and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_der2asn1_encprivpart ()

```
Shishi_asn1
shishi_der2asn1_encprivpart (Shishi *handle,
                             const char *der,
                             size_t derlen);
```

Decode DER encoding of EncKrbPrivPart and create a ASN.1 structure.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| der | input character array with DER encoding. | |
| derlen | length of input character array with DER encoding. | |

#### Returns

Returns ASN.1 structure corresponding to DER data.

### shishi_ap ()

```
int
shishi_ap (Shishi *handle,
           Shishi_ap **ap);
```

Create a new AP exchange with a random subkey of the default encryption type from configuration. Note that there is no guarantee that the receiver will understand that key type, you should probably use shishi_ap_etype() or shishi_ap_nosubkey() instead. In the future, this function will likely behave as shishi_ap_nosubkey() and shishi_ap_nosubkey() will be removed.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| ap | pointer to new structure that holds information about AP exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_etype ()

```
int
shishi_ap_etype (Shishi *handle,
                 Shishi_ap **ap,
                 int etype);
```

Create a new AP exchange with a random subkey of indicated encryption type.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| ap | pointer to new structure that holds information about AP exchange |
| etype | encryption type of newly generated random subkey. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_nosubkey ()

```
int
shishi_ap_nosubkey (Shishi *handle,
                    Shishi_ap **ap);
```

Create a new AP exchange without subkey in authenticator.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

| | |
|---|---|
| ap | pointer to new structure that holds information about AP exchange |

### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_done ()

```
void
shishi_ap_done (Shishi_ap *ap);
```

Deallocate resources associated with AP exchange. This should be called by the application when it no longer need to utilize the AP exchange handle.

### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |

### shishi_ap_set_tktoptions ()

```
int
shishi_ap_set_tktoptions (Shishi_ap *ap,
                          Shishi_tkt *tkt,
                          int options);
```

Set the ticket (see shishi_ap_tkt_set()) and set the AP-REQ apoptions (see shishi_apreq_options_set()).

### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |
| tkt | ticket to set in AP. |
| options | AP-REQ options to set in AP. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_tktoptions ()

```
int
shishi_ap_tktoptions (Shishi *handle,
                      Shishi_ap **ap,
                      Shishi_tkt *tkt,
                      int options);
```

Create a new AP exchange using shishi_ap(), and set the ticket and AP-REQ apoptions using shishi_ap_set_tktoptions(). A random session key is added to the authenticator, using the same keytype as the ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ap | pointer to new structure that holds information about AP exchange | |
| tkt | ticket to set in newly created AP. | |
| options | AP-REQ options to set in newly created AP. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_etype_tktoptionsdata ()

```
int
shishi_ap_etype_tktoptionsdata (Shishi *handle,
                                Shishi_ap **ap,
                                int32_t etype,
                                Shishi_tkt *tkt,
                                int options,
                                const char *data,
                                size_t len);
```

Create a new AP exchange using shishi_ap(), and set the ticket, AP-REQ apoptions and the Authenticator checksum data using shishi_ap_set_tktoptionsdata(). A random session key is added to the authenticator, using the same keytype as the ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ap | pointer to new structure that holds information about AP exchange | |
| etype | encryption type of newly generated random subkey. | |
| tkt | ticket to set in newly created AP. | |
| options | AP-REQ options to set in newly created AP. | |
| data | input array with data to checksum in Authenticator. | |
| len | length of input array with data to checksum in Authenticator. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_set_tktoptionsdata ()

```
int
shishi_ap_set_tktoptionsdata (Shishi_ap *ap,
                              Shishi_tkt *tkt,
                              int options,
                              const char *data,
                              size_t len);
```

Set the ticket (see shishi_ap_tkt_set()) and set the AP-REQ apoptions (see shishi_apreq_options_set()) and set the Authenticator checksum data.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| tkt | ticket to set in AP. | |
| options | AP-REQ options to set in AP. | |
| data | input array with data to checksum in Authenticator. | |
| len | length of input array with data to checksum in Authenticator. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_tktoptionsdata ()

```
int
shishi_ap_tktoptionsdata (Shishi *handle,
                          Shishi_ap **ap,
                          Shishi_tkt *tkt,
                          int options,
                          const char *data,
                          size_t len);
```

Create a new AP exchange using shishi_ap(), and set the ticket, AP-REQ apoptions and the Authenticator checksum data using shishi_ap_set_tktoptionsdata(). A random session key is added to the authenticator, using the same keytype as the ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ap | pointer to new structure that holds information about AP exchange | |

| | | |
|---|---|---|
| tkt | ticket to set in newly created AP. | |
| options | AP-REQ options to set in newly created AP. | |
| data | input array with data to checksum in Authenticator. | |
| len | length of input array with data to checksum in Authenticator. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_set_tktoptionsraw ()

```
int
shishi_ap_set_tktoptionsraw (Shishi_ap *ap,
                             Shishi_tkt *tkt,
                             int options,
                             int32_t cksumtype,
                             const char *data,
                             size_t len);
```

Set the ticket (see shishi_ap_tkt_set()) and set the AP-REQ apoptions (see shishi_apreq_options_set()) and set the raw Authenticator checksum data.

### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| tkt | ticket to set in AP. | |
| options | AP-REQ options to set in AP. | |
| cksumtype | authenticator checksum type to set in AP. | |
| data | input array with data to store in checksum field in Authenticator. | |
| len | length of input array with data to store in checksum field in Authenticator. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_tktoptionsraw ()

```
int
shishi_ap_tktoptionsraw (Shishi *handle,
                         Shishi_ap **ap,
                         Shishi_tkt *tkt,
                         int options,
                         int32_t cksumtype,
                         const char *data,
                         size_t len);
```

Create a new AP exchange using shishi_ap(), and set the ticket, AP-REQ apoptions and the raw Authenticator checksum data field using shishi_ap_set_tktoptionsraw(). A random session key is added to the authenticator, using the same keytype as the ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ap | pointer to new structure that holds information about AP exchange | |
| tkt | ticket to set in newly created AP. | |
| options | AP-REQ options to set in newly created AP. | |
| cksumtype | authenticator checksum type to set in AP. | |
| data | input array with data to store in checksum field in Authenticator. | |
| len | length of input array with data to store in checksum field in Authenticator. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_set_tktoptionsasn1usage ()

```
int
shishi_ap_set_tktoptionsasn1usage (Shishi_ap *ap,
                                   Shishi_tkt *tkt,
                                   int options,
                                   Shishi_asn1 node,
                                   const char *field,
                                   int authenticatorcksumkeyusage,
                                   int authenticatorkeyusage);
```

Set ticket, options and authenticator checksum data using shishi_ap_set_tktoptionsdata(). The authenticator checksum data is the DER encoding of the ASN.1 field provided.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| tkt | ticket to set in AP. | |
| options | AP-REQ options to set in AP. | |

| | | |
|---|---|---|
| node | input ASN.1 structure to store as authenticator checksum data. | |
| field | field in ASN.1 structure to use. | |
| authenticatorcksumkeyusage | key usage for checksum in authenticator. | |
| authenticatorkeyusage | key usage for authenticator. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_tktoptionsasn1usage ()

```
int
shishi_ap_tktoptionsasn1usage (Shishi *handle,
                               Shishi_ap **ap,
                               Shishi_tkt *tkt,
                               int options,
                               Shishi_asn1 node,
                               const char *field,
                               int authenticatorcksumkeyusage,
                               int authenticatorkeyusage);
```

Create a new AP exchange using shishi_ap(), and set ticket, options and authenticator checksum data from the DER encoding of the ASN.1 field using shishi_ap_set_tktoptionsasn1usage(). A random session key is added to the authenticator, using the same keytype as the ticket.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| ap | pointer to new structure that holds information about AP exchange | |
| tkt | ticket to set in newly created AP. | |
| options | AP-REQ options to set in newly created AP. | |
| node | input ASN.1 structure to store as authenticator checksum data. | |
| field | field in ASN.1 structure to use. | |
| authenticatorcksumkeyusage | key usage for checksum in authenticator. | |
| authenticatorkeyusage | key usage for authenticator. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_tkt ()

```
Shishi_tkt~*
shishi_ap_tkt (Shishi_ap *ap);
```

Get Ticket from AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |

#### Returns

Returns the ticket from the AP exchange, or NULL if not yet set or an error occured.

### shishi_ap_tkt_set ()

```
void
shishi_ap_tkt_set (Shishi_ap *ap,
                   Shishi_tkt *tkt);
```

Set the Ticket in the AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| tkt | ticket to store in AP. | |

### shishi_ap_authenticator_cksumdata ()

```
int
shishi_ap_authenticator_cksumdata (Shishi_ap *ap,
                                   char *out,
                                   size_t *len);
```

Get checksum data from Authenticator.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| out | output array that holds authenticator checksum data. | |

| | |
|---|---|
| len | on input, maximum length of output array that holds authenticator checksum data, on output actual length of output array that holds authenticator checksum data. |

**Returns**

Returns SHISHI_OK if successful, or SHISHI_TOO_SMALL_BUFFER if buffer provided was too small (then *len* will hold necessary buffer size).

### shishi_ap_authenticator_cksumdata_set ()

```
void
shishi_ap_authenticator_cksumdata_set (Shishi_ap *ap,
                                       const char *authenticatorcksumdata,
                                       size_t authenticatorcksumdatalen);
```

Set the Authenticator Checksum Data in the AP exchange. This is the data that will be checksumed, and the checksum placed in the checksum field. It is not the actual checksum field. See also shishi_ap_authenticator_cksumraw_set.

**Parameters**

| | |
|---|---|
| ap | structure that holds information about AP exchange |
| authenticatorcksumdata | input array with data to compute checksum on and store in Authenticator in AP-REQ. |
| authenticatorcksumdatalen | length of input array with data to compute checksum on and store in Authenticator in AP-REQ. |

### shishi_ap_authenticator_cksumraw_set ()

```
void
shishi_ap_authenticator_cksumraw_set (Shishi_ap *ap,
                                      int32_t authenticatorcksumtype,
                                      const char *authenticatorcksumraw,
                                      size_t authenticatorcksumrawlen);
```

Set the Authenticator Checksum Data in the AP exchange. This is the actual checksum field, not data to compute checksum on and then store in the checksum field. See also shishi_ap_authenticator_cksumdata_set.

**Parameters**

| | |
|---|---|
| ap | structure that holds information about AP exchange |

| | | |
|---|---|---|
| authenticatorcksumtype | authenticator checksum type to set in AP. | |
| authenticatorcksumraw | input array with authenticator checksum field value to set in Authenticator in AP-REQ. | |
| authenticatorcksumrawlen | length of input array with authenticator checksum field value to set in Authenticator in AP-REQ. | |

### shishi_ap_authenticator_cksumtype ()

```
int32_t
shishi_ap_authenticator_cksumtype (Shishi_ap *ap);
```

Get the Authenticator Checksum Type in the AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |

#### Returns

Return the authenticator checksum type.

### shishi_ap_authenticator_cksumtype_set ()

```
void
shishi_ap_authenticator_cksumtype_set (Shishi_ap *ap,
                                       int32_t cksumtype);
```

Set the Authenticator Checksum Type in the AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| cksumtype | authenticator checksum type to set in AP. | |

### shishi_ap_authenticator ()

```
Shishi_asn1
shishi_ap_authenticator (Shishi_ap *ap);
```

Get ASN.1 Authenticator structure from AP exchange.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |

#### Returns

Returns the Authenticator from the AP exchange, or NULL if not yet set or an error occured.

### shishi_ap_authenticator_set ()

```
void
shishi_ap_authenticator_set (Shishi_ap *ap,
                             Shishi_asn1 authenticator);
```

Set the Authenticator in the AP exchange.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |
| authenticator | authenticator to store in AP. |

### shishi_ap_req ()

```
Shishi_asn1
shishi_ap_req (Shishi_ap *ap);
```

Get ASN.1 AP-REQ structure from AP exchange.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |

#### Returns

Returns the AP-REQ from the AP exchange, or NULL if not yet set or an error occured.

### shishi_ap_req_set ()

```
void
shishi_ap_req_set (Shishi_ap *ap,
                   Shishi_asn1 apreq);
```

Set the AP-REQ in the AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| apreq | apreq to store in AP. | |

### shishi_ap_req_der ()

```
int
shishi_ap_req_der (Shishi_ap *ap,
                   char **out,
                   size_t *outlen);
```

Build AP-REQ using shishi_ap_req_build() and DER encode it. `out` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| out | pointer to output array with der encoding of AP-REQ. | |
| outlen | pointer to length of output array with der encoding of AP-REQ. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_req_der_set ()

```
int
shishi_ap_req_der_set (Shishi_ap *ap,
                       char *der,
                       size_t derlen);
```

DER decode AP-REQ and set it AP exchange. If decoding fails, the AP-REQ in the AP exchange is lost.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| der | input array with DER encoded AP-REQ. | |
| derlen | length of input array with DER encoded AP-REQ. | |

#### Returns

Returns SHISHI_OK.

### shishi_ap_req_build ()

```
int
shishi_ap_req_build (Shishi_ap *ap);
```

Checksum data in authenticator and add ticket and authenticator to AP-REQ.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_req_asn1 ()

```
int
shishi_ap_req_asn1 (Shishi_ap *ap,
                    Shishi_asn1 *apreq);
```

Build AP-REQ using shishi_ap_req_build() and return it.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |
| apreq | output AP-REQ variable. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_key ()

```
Shishi_key~*
shishi_ap_key (Shishi_ap *ap);
```

Extract the application key from AP. If subkeys are used, it is taken from the Authenticator, otherwise the session key is used.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |

#### Returns

Return application key from AP.

### shishi_ap_req_decode ()

```
int
shishi_ap_req_decode (Shishi_ap *ap);
```

Decode ticket in AP-REQ and set the Ticket fields in the AP exchange.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_req_process ()

```
int
shishi_ap_req_process (Shishi_ap *ap,
                       Shishi_key *key);
```

Decrypt ticket in AP-REQ using supplied key and decrypt Authenticator in AP-REQ using key in decrypted ticket, and on success set the Ticket and Authenticator fields in the AP exchange.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |
| key | cryptographic key used to decrypt ticket in AP-REQ. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_req_process_keyusage ()

```
int
shishi_ap_req_process_keyusage (Shishi_ap *ap,
                                Shishi_key *key,
                                int32_t keyusage);
```

Decrypt ticket in AP-REQ using supplied key and decrypt Authenticator in AP-REQ using key in decrypted ticket, and on success set the Ticket and Authenticator fields in the AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| key | cryptographic key used to decrypt ticket in AP-REQ. | |
| keyusage | key usage to use during decryption, for normal AP-REQ's this is normally SHISHI_KEYUSAGE_APREQ_AUTHENTICATOR, for AP-REQ's part of TGS-REQ's, this is normally SHISHI_KEYUSAGE_TGSREQ_APREQ_AUTHENTICATOR. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_rep ()

```
Shishi_asn1
shishi_ap_rep (Shishi_ap *ap);
```

Get ASN.1 AP-REP structure from AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |

#### Returns

Returns the AP-REP from the AP exchange, or NULL if not yet set or an error occured.

### shishi_ap_rep_set ()

```
void
shishi_ap_rep_set (Shishi_ap *ap,
                   Shishi_asn1 aprep);
```

Set the AP-REP in the AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| aprep | aprep to store in AP. | |

### shishi_ap_rep_der ()

```
int
shishi_ap_rep_der (Shishi_ap *ap,
                   char **out,
                   size_t *outlen);
```

Build AP-REP using shishi_ap_rep_build() and DER encode it. `out` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| out | output array with newly allocated DER encoding of AP-REP. | |
| outlen | length of output array with DER encoding of AP-REP. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_rep_der_set ()

```
int
shishi_ap_rep_der_set (Shishi_ap *ap,
                       char *der,
                       size_t derlen);
```

DER decode AP-REP and set it AP exchange. If decoding fails, the AP-REP in the AP exchange remains.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| der | input array with DER encoded AP-REP. | |
| derlen | length of input array with DER encoded AP-REP. | |

#### Returns

Returns SHISHI_OK.

### shishi_ap_rep_verify ()

```
int
shishi_ap_rep_verify (Shishi_ap *ap);
```

Verify AP-REP compared to Authenticator.

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |

#### Returns

Returns SHISHI_OK, SHISHI_APREP_VERIFY_FAILED or an error.

### shishi_ap_rep_verify_der ()

```
int
shishi_ap_rep_verify_der (Shishi_ap *ap,
                          char *der,
                          size_t derlen);
```

DER decode AP-REP and set it in AP exchange using shishi_ap_rep_der_set() and verify it using shishi_ap_rep_verify().

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |
| der | input array with DER encoded AP-REP. |
| derlen | length of input array with DER encoded AP-REP. |

#### Returns

Returns SHISHI_OK, SHISHI_APREP_VERIFY_FAILED or an error.

### shishi_ap_rep_verify_asn1 ()

```
int
shishi_ap_rep_verify_asn1 (Shishi_ap *ap,
                           Shishi_asn1 aprep);
```

Set the AP-REP in the AP exchange using shishi_ap_rep_set() and verify it using shishi_ap_rep_verify().

#### Parameters

| | |
|---|---|
| ap | structure that holds information about AP exchange |
| aprep | input AP-REP. |

#### Returns

Returns SHISHI_OK, SHISHI_APREP_VERIFY_FAILED or an error.

### shishi_ap_rep_asn1 ()

```
int
shishi_ap_rep_asn1 (Shishi_ap *ap,
                    Shishi_asn1 *aprep);
```

Build AP-REP using shishi_ap_rep_build() and return it.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| aprep | output AP-REP variable. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_rep_build ()

```
int
shishi_ap_rep_build (Shishi_ap *ap);
```

Checksum data in authenticator and add ticket and authenticator to AP-REP.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_ap_encapreppart ()

```
Shishi_asn1
shishi_ap_encapreppart (Shishi_ap *ap);
```

Get ASN.1 EncAPRepPart structure from AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |

#### Returns

Returns the EncAPREPPart from the AP exchange, or NULL if not yet set or an error occured.

### shishi_ap_encapreppart_set ()

```
void
shishi_ap_encapreppart_set (Shishi_ap *ap,
                            Shishi_asn1 encapreppart);
```

Set the EncAPRepPart in the AP exchange.

#### Parameters

| | | |
|---|---|---|
| ap | structure that holds information about AP exchange | |
| encapreppart | EncAPRepPart to store in AP. | |

### shishi_ap_option2string ()

```
const char~*
shishi_ap_option2string (Shishi_apoptions option);
```

Convert AP-Option type to AP-Option name string. Note that `option` must be just one of the AP-Option types, it cannot be an binary ORed indicating several AP-Options.

#### Parameters

| | | |
|---|---|---|
| option | enumerated AP-Option type, see Shishi_apoptions. | |

#### Returns

Returns static string with name of AP-Option that must not be deallocated, or "unknown" if AP-Option was not understood.

### shishi_ap_string2option ()

```
Shishi_apoptions
shishi_ap_string2option (const char *str);
```

Convert AP-Option name to AP-Option type.

#### Parameters

| | | |
|---|---|---|
| str | zero terminated character array with name of AP-Option, e.g. "use-session-key". | |

#### Returns

Returns enumerated type member corresponding to AP-Option, or 0 if string was not understood.

### shishi_key_principal ()

```
const char~*
shishi_key_principal (const Shishi_key *key);
```

Get the principal part of the key owner principal name, i.e., except the realm.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |

#### Returns

Returns the principal owning the key. (Not a copy of it, so don't modify or deallocate it.)

### shishi_key_principal_set ()

```
void
shishi_key_principal_set (Shishi_key *key,
                          const char *principal);
```

Set the principal owning the key. The string is copied into the key, so you can dispose of the variable immediately after calling this function.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |
| principal | string with new principal name. |

### shishi_key_realm ()

```
const char~*
shishi_key_realm (const Shishi_key *key);
```

Get the realm part of the key owner principal name.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |

#### Returns

Returns the realm for the principal owning the key. (Not a copy of it, so don't modify or deallocate it.)

### shishi_key_realm_set ()

```
void
shishi_key_realm_set (Shishi_key *key,
                      const char *realm);
```

Set the realm for the principal owning the key. The string is copied into the key, so you can dispose of the variable immediately after calling this function.

#### Parameters

| | | |
|---|---|---|
| key | structure that holds key information | |
| realm | string with new realm name. | |

### shishi_key_type ()

```
int
shishi_key_type (const Shishi_key *key);
```

Get key type.

#### Parameters

| | | |
|---|---|---|
| key | structure that holds key information | |

#### Returns

Returns the type of key as an integer as described in the standard.

### shishi_key_type_set ()

```
void
shishi_key_type_set (Shishi_key *key,
                     int32_t type);
```

Set the type of key in key structure.

#### Parameters

| | | |
|---|---|---|
| key | structure that holds key information | |
| type | type to set in key. | |

### shishi_key_value ()

```
const char~*
shishi_key_value (const Shishi_key *key);
```

Get the raw key bytes.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |

#### Returns

Returns the key value as a pointer which is valid throughout the lifetime of the key structure.

### shishi_key_value_set ()

```
void
shishi_key_value_set (Shishi_key *key,
                      const char *value);
```

Set the key value and length in key structure. The value is copied into the key (in other words, you can deallocate `value` right after calling this function without modifying the value inside the key).

#### Parameters

| | |
|---|---|
| key | structure that holds key information |
| value | input array with key data. |

### shishi_key_name ()

```
const char~*
shishi_key_name (Shishi_key *key);
```

Calls shishi_cipher_name for key type.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |

#### Returns

Return name of key.

### shishi_key_length ()

```
size_t
shishi_key_length (const Shishi_key *key);
```

Calls shishi_cipher_keylen for key type.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |

#### Returns

Returns the length of the key value.

### shishi_key_version ()

```
uint32_t
shishi_key_version (const Shishi_key *key);
```

Get the "kvno" (key version) of key. It will be UINT32_MAX if the key is not long-lived.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |

#### Returns

Returns the version of key ("kvno").

### shishi_key_version_set ()

```
void
shishi_key_version_set (Shishi_key *key,
                        uint32_t kvno);
```

Set the version of key ("kvno") in key structure. Use UINT32_MAX for non-ptermanent keys.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |
| kvno | new version integer. |

### shishi_key_timestamp ()

```
time_t
shishi_key_timestamp (const Shishi_key *key);
```

Get the time the key was established. Typically only present when the key was imported from a keytab format.

#### Parameters

| | |
|---|---|
| key | structure that holds key information |

#### Returns

Returns the time the key was established, or (time_t)-1 if not available.

Since: 0.0.42

### shishi_key_timestamp_set ()

```
void
shishi_key_timestamp_set (Shishi_key *key,
                          time_t timestamp);
```

Set the time the key was established. Typically only relevant when exporting the key to keytab format.

#### Parameters

| | | |
|---|---|---|
| key | structure that holds key information | |
| timestamp | new timestamp. | |

Since: 0.0.42

### shishi_key ()

```
int
shishi_key (Shishi *handle,
            Shishi_key **key);
```

Create a new Key information structure.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| key | pointer to structure that will hold newly created key information | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_key_done ()

```
void
shishi_key_done (Shishi_key *key);
```

Deallocates key information structure.

#### Parameters

| | |
|---|---|
| key | pointer to structure that holds key information. |

### shishi_key_copy ()

```
void
shishi_key_copy (Shishi_key *dstkey,
                 Shishi_key *srckey);
```

Copies source key into existing allocated destination key.

#### Parameters

| | |
|---|---|
| dstkey | structure that holds destination key information |
| srckey | structure that holds source key information |

### shishi_key_print ()

```
int
shishi_key_print (Shishi *handle,
                  FILE *fh,
                  const Shishi_key *key);
```

Prints an ASCII representation of a key structure `key` to the file descriptor `fh` . Example output:

-----BEGIN SHISHI KEY----- Keytype: 18 (aes256-cts-hmac-sha1-96) Principal: host/latte.josefsson.org Realm: JOSEFSSON.ORG Key-Version-Number: 1 Timestamp: 20130420150337Z

P1QdeW/oSiag/bTyVEBAY2msiGSTmgLXlopuCKoppDs= -----END SHISHI KEY-----

#### Parameters

| | |
|---|---|
| handle | Shishi handle as allocated by shishi_init(). |
| fh | File handle open for writing. |
| key | Key to print. |

#### Returns

Returns SHISHI_OK if successful. The only failure is SHISHI_MALLOC_ERROR.

### shishi_key_to_file ()

```
int
shishi_key_to_file (Shishi *handle,
                    const char *filename,
                    Shishi_key *key);
```

Prints an ASCII representation of a key structure `key` to the file `filename` . The text is appended if the file exists. See shishi_key_print() for an example of output text.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi handle as allocated by shishi_init(). | |
| filename | Name of file, to which the key text is appended. | |
| key | Key to print. | |

#### Returns

Returns SHISHI_OK if successful. Failures are due to I/O issues, or to allocation.

### shishi_key_parse ()

```
int
shishi_key_parse (Shishi *handle,
                  FILE *fh,
                  Shishi_key **key);
```

### shishi_key_random ()

```
int
shishi_key_random (Shishi *handle,
                   int32_t type,
                   Shishi_key **key);
```

Create a new Key information structure for the key type and some random data. KEY contains a newly allocated structure only if this function is successful.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| type | type of key. | |
| key | pointer to structure that will hold newly created key information | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_key_from_value ()

```
int
shishi_key_from_value (Shishi *handle,
                       int32_t type,
                       const char *value,
                       Shishi_key **key);
```

Create a new Key information structure, and set the key type and key value. KEY contains a newly allocated structure only if this function is successful.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| type | type of key. | |
| value | input array with key value, or NULL. | |
| key | pointer to structure that will hold newly created key information | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_key_from_base64 ()

```
int
shishi_key_from_base64 (Shishi *handle,
                        int32_t type,
                        const char *value,
                        Shishi_key **key);
```

Create a new Key information structure, and set the key type and key value. KEY contains a newly allocated structure only if this function is successful.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| type | type of key. | |
| value | input string with base64 encoded key value, or NULL. | |
| key | pointer to structure that will hold newly created key information | |

#### Returns

Returns SHISHI_INVALID_KEY if the base64 encoded key length doesn't match the key type, and SHISHI_OK on success.

### shishi_key_from_random ()

```
int
shishi_key_from_random (Shishi *handle,
                        int32_t type,
                        const char *rnd,
                        size_t rndlen,
                        Shishi_key **outkey);
```

Create a new Key information structure, and set the key type and key value using shishi_random_to_key(). KEY contains a newly allocated structure only if this function is successful.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| type | type of key. | |
| rnd | random data. | |
| rndlen | length of random data. | |
| outkey | pointer to structure that will hold newly created key information | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_key_from_string ()

```
int
shishi_key_from_string (Shishi *handle,
                        int32_t type,
                        const char *password,
                        size_t passwordlen,
                        const char *salt,
                        size_t saltlen,
                        const char *parameter,
                        Shishi_key **outkey);
```

Create a new Key information structure, and set the key type and key value using shishi_string_to_key(). KEY contains a newly allocated structure only if this function is successful.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| type | type of key. | |
| password | input array containing password. | |
| passwordlen | length of input array containing password. | |
| salt | input array containing salt. | |
| saltlen | length of input array containing salt. | |
| parameter | input array with opaque encryption type specific information. | |
| outkey | pointer to structure that will hold newly created key information | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_key_from_name ()

```
int
shishi_key_from_name (Shishi *handle,
                      int32_t type,
                      const char *name,
                      const char *password,
                      size_t passwordlen,
                      const char *parameter,
                      Shishi_key **outkey);
```

Create a new Key information structure, and derive the key from principal name and password using shishi_key_from_name(). The salt is derived from the principal name by concatenating the decoded realm and principal.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| type | type of key. | |
| name | principal name of user. | |
| password | input array containing password. | |
| passwordlen | length of input array containing password. | |
| parameter | input array with opaque encryption type specific information. | |
| outkey | pointer to structure that will hold newly created key information | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_keys ()

```
int
shishi_keys (Shishi *handle,
             Shishi_keys **keys);
```

Get a new key set handle.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| keys | output pointer to newly allocated keys handle. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_keys_done ()

```
void
shishi_keys_done (Shishi_keys **keys);
```

Deallocates all resources associated with key set. The key set handle must not be used in calls to other shishi_keys_*() functions after this.

#### Parameters

| | |
|---|---|
| keys | key set handle as allocated by shishi_keys(). |

### shishi_keys_size ()

```
int
shishi_keys_size (Shishi_keys *keys);
```

Get size of key set.

#### Parameters

| | |
|---|---|
| keys | key set handle as allocated by shishi_keys(). |

#### Returns

Returns number of keys stored in key set.

### shishi_keys_nth ()

```
const Shishi_key~*
shishi_keys_nth (Shishi_keys *keys,
                 int keyno);
```

Get the n:th ticket in key set.

#### Parameters

| | |
|---|---|
| keys | key set handle as allocated by shishi_keys(). |
| keyno | integer indicating requested key in key set. |

#### Returns

Returns a key handle to the keyno:th key in the key set, or NULL if *keys* is invalid or *keyno* is out of bounds. The first key is *keyno* 0, the second key *keyno* 1, and so on.

### shishi_keys_remove ()

```
void
shishi_keys_remove (Shishi_keys *keys,
                    int keyno);
```

Remove a key, indexed by *keyno* , in given key set.

#### Parameters

| | |
|---|---|
| keys | key set handle as allocated by shishi_keys(). |
| keyno | key number of key in the set to remove. The first key is key number 0. |

### shishi_keys_add ()

```
int
shishi_keys_add (Shishi_keys *keys,
                 Shishi_key *key);
```

Add a key to the key set. A deep copy of the key is stored, so changing *key* , or deallocating it, will not modify the value stored in the key set.

#### Parameters

| | |
|---|---|
| keys | key set handle as allocated by shishi_keys(). |
| key | key to be added to key set. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_keys_print ()

```
int
shishi_keys_print (Shishi_keys *keys,
                   FILE *fh);
```

Print all keys in set using shishi_key_print.

#### Parameters

| | |
|---|---|
| keys | key set to print. |
| fh | file handle, open for writing, to print keys to. |

#### Returns

Returns SHISHI_OK on success.

### shishi_keys_from_file ()

```
int
shishi_keys_from_file (Shishi_keys *keys,
                       const char *filename);
```

Read zero or more keys from file `filename` and append them to the keyset `keys` . See shishi_key_print() for the format of the input.

#### Parameters

| | | |
|---|---|---|
| keys | key set handle as allocated by shishi_keys(). | |
| filename | filename to read keys from. | |

#### Returns

Returns SHISHI_OK iff successful.

Since: 0.0.42

### shishi_keys_to_file ()

```
int
shishi_keys_to_file (Shishi *handle,
                     const char *filename,
                     Shishi_keys *keys);
```

Print an ASCII representation of a key structure to a file, for each key in the key set. The file is appended to if it exists. See shishi_key_print() for the format of the output.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| filename | filename to append key to. | |
| keys | set of keys to print. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_keys_for_serverrealm_in_file ()

```
Shishi_key~*
shishi_keys_for_serverrealm_in_file (Shishi *handle,
                                     const char *filename,
                                     const char *server,
                                     const char *realm);
```

Get keys that match specified `server` and `realm` from the key set file `filename` .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| filename | file to read keys from. | |
| server | server name to get key for. | |
| realm | realm of server to get key for. | |

#### Returns

Returns the key for specific server and realm, read from the indicated file, or NULL if no key could be found or an error encountered.

### shishi_keys_for_server_in_file ()

```
Shishi_key~*
shishi_keys_for_server_in_file (Shishi *handle,
                                const char *filename,
                                const char *server);
```

Get key for specified `server` from `filename` .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| filename | file to read keys from. | |
| server | server name to get key for. | |

#### Returns

Returns the key for specific server, read from the indicated file, or NULL if no key could be found or an error encountered.

### shishi_keys_for_localservicerealm_in_file ()

```
Shishi_key~*
shishi_keys_for_localservicerealm_in_file
                               (Shishi *handle,
                                const char *filename,
                                const char *service,
                                const char *realm);
```

Get key for specified `service` and `realm` from `filename` .

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| filename | file to read keys from. | |
| service | service to get key for. | |
| realm | realm of server to get key for, or NULL for default realm. | |

#### Returns

Returns the key for the server "SERVICE/HOSTNAME*REALM* " (where HOSTNAME is the current system's hostname), read from the default host keys file (see shishi_hostkeys_default_file()), or NULL if no key could be found or an error encountered.

### shishi_keys_add_keytab_mem ()

```
int
shishi_keys_add_keytab_mem (Shishi *handle,
                            const char *data,
                            size_t len,
                            Shishi_keys *keys);
```

Read keys from a MIT keytab data structure, and add them to the key set.

The format of keytab's is proprietary, and this function support the 0x0501 and 0x0502 formats. See the section The MIT Kerberos Keytab Binary File Format in the Shishi manual for a description of the reverse-engineered format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| data | constant memory buffer with keytab of *len* size. | |
| len | size of memory buffer with keytab data. | |
| keys | allocated key set to store keys in. | |

#### Returns

Returns SHISHI_KEYTAB_ERROR if the data does not represent a valid keytab structure, and SHISHI_OK on success.

### shishi_keys_add_keytab_file ()

```
int
shishi_keys_add_keytab_file (Shishi *handle,
                             const char *filename,
                             Shishi_keys *keys);
```

Read keys from a MIT keytab data structure from a file, and add the keys to the key set.

The format of keytab's is proprietary, and this function support the 0x0501 and 0x0502 formats. See the section The MIT Kerberos Keytab Binary File Format in the Shishi manual for a description of the reverse-engineered format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| filename | name of file to read. | |
| keys | allocated key set to store keys in. | |

#### Returns

Returns SHISHI_IO_ERROR if the file cannot be read, SHISHI_KEYTAB_ERROR if the data cannot be parsed as a valid keytab structure, and SHISHI_OK on success.

### shishi_keys_from_keytab_mem ()

```
int
shishi_keys_from_keytab_mem (Shishi *handle,
                             const char *data,
                             size_t len,
                             Shishi_keys **outkeys);
```

Create a new key set populated with keys from a MIT keytab data structure read from a memory block.

The format of keytab's is proprietary, and this function support the 0x0501 and 0x0502 formats. See the section The MIT Kerberos Keytab Binary File Format in the Shishi manual for a description of the reverse-engineered format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| data | constant memory buffer with keytab of `len` size. | |
| len | size of memory buffer with keytab data. | |
| outkeys | pointer to key set that will be allocated and populated, must be deallocated by caller on succes. | |

#### Returns

Returns SHISHI_KEYTAB_ERROR if the data does not represent a valid keytab structure, and SHISHI_OK on success.

### shishi_keys_from_keytab_file ()

```
int
shishi_keys_from_keytab_file (Shishi *handle,
                              const char *filename,
                              Shishi_keys **outkeys);
```

Create a new key set populated with keys from a MIT keytab data structure read from a file.

The format of keytab's is proprietary, and this function support the 0x0501 and 0x0502 formats. See the section The MIT Kerberos Keytab Binary File Format in the Shishi manual for a description of the reverse-engineered format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| filename | name of file to read. | |
| outkeys | pointer to key set that will be allocated and populated, must be deallocated by caller on succes. | |

#### Returns

Returns SHISHI_IO_ERROR if the file cannot be read, SHISHI_KEYTAB_ERROR if the data cannot be parsed as a valid keytab structure, and SHISHI_OK on success.

### shishi_keys_to_keytab_mem ()

```
int
shishi_keys_to_keytab_mem (Shishi *handle,
                           Shishi_keys *keys,
                           char **out,
                           size_t *len);
```

Write keys to a MIT keytab data structure.

The format of keytab's is proprietary, and this function writes the 0x0502 format. See the section The MIT Kerberos Keytab Binary File Format in the Shishi manual for a description of the reverse-engineered format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| keys | key set to convert to keytab format. | |
| out | constant memory buffer with keytab of `len` size. | |
| len | size of memory buffer with keytab data. | |

#### Returns

On success SHISHI_OK is returned, otherwise an error code.

Since: 0.0.42

### shishi_keys_to_keytab_file ()

```
int
shishi_keys_to_keytab_file (Shishi *handle,
                            Shishi_keys *keys,
                            const char *filename);
```

Write keys to a MIT keytab data structure.

The format of keytab's is proprietary, and this function writes the 0x0502 format. See the section The MIT Kerberos Keytab Binary File Format in the Shishi manual for a description of the reverse-engineered format.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| keys | keyset to write. | |
| filename | name of file to write. | |

#### Returns

SHISHI_FOPEN_ERROR if there is a problem opening `filename` for writing, SHISHI_IO_ERROR if there is problem writing the file, and SHISHI_OK on success.

Since: 0.0.42

### shishi_hostkeys_default_file ()

```
const char~*
shishi_hostkeys_default_file (Shishi *handle);
```

Get file name of default host key file.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Returns the default host key filename used in the library. (Not a copy of it, so don't modify or deallocate it.)

### shishi_hostkeys_default_file_set ()

```
void
shishi_hostkeys_default_file_set (Shishi *handle,
                                  const char *hostkeysfile);
```

Set the default host key filename used in the library. The string is copied into the library, so you can dispose of the variable immediately after calling this function.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| hostkeysfile | string with new default hostkeys file name, or NULL to reset to default. |

### shishi_hostkeys_for_server ()

```
Shishi_key~*
shishi_hostkeys_for_server (Shishi *handle,
                            const char *server);
```

Get host key for `server` .

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| server | server name to get key for |

### Returns

Returns the key for specific server, read from the default host keys file (see shishi_hostkeys_default_file()), or NULL if no key could be found or an error encountered.

### shishi_hostkeys_for_serverrealm ()

```
Shishi_key~*
shishi_hostkeys_for_serverrealm (Shishi *handle,
                                 const char *server,
                                 const char *realm);
```

Get host key for `server` in `realm` .

### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| server | server name to get key for |
| realm | realm of server to get key for. |

### Returns

Returns the key for specific server and realm, read from the default host keys file (see shishi_hostkeys_default_file()), or NULL if no key could be found or an error encountered.

### shishi_hostkeys_for_localservicerealm ()

```
Shishi_key~*
shishi_hostkeys_for_localservicerealm (Shishi *handle,
                                       const char *service,
                                       const char *realm);
```

Get host key for `service` on current host in `realm` .

### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| service | service to get key for. |
| realm | realm of server to get key for, or NULL for default realm. |

### Returns

Returns the key for the server "SERVICE/HOSTNAME`REALM` " (where HOSTNAME is the current system's hostname), read from the default host keys file (see shishi_hostkeys_default_file()), or NULL if no key could be found or an error encountered.

### shishi_hostkeys_for_localservice ()

```
Shishi_key~*
shishi_hostkeys_for_localservice (Shishi *handle,
                                  const char *service);
```

Get host key for `service` on current host in default realm.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| service | service to get key for. |

#### Returns

Returns the key for the server "SERVICE/HOSTNAME" (where HOSTNAME is the current system's hostname), read from the default host keys file (see shishi_hostkeys_default_file()), or NULL if no key could be found or an error encountered.

### shishi_encapreppart ()

```
Shishi_asn1
shishi_encapreppart (Shishi *handle);
```

This function creates a new EncAPRepPart, populated with some default values. It uses the current time as returned by the system for the ctime and cusec fields.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the encapreppart or NULL on failure.

### shishi_encapreppart_time_copy ()

```
int
shishi_encapreppart_time_copy (Shishi *handle,
                               Shishi_asn1 encapreppart,
                               Shishi_asn1 authenticator);
```

Copy time fields from Authenticator into EncAPRepPart.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encapreppart | EncAPRepPart as allocated by shishi_encapreppart(). |
| authenticator | Authenticator to copy time fields from. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_ctime ()

```
int
shishi_encapreppart_ctime (Shishi *handle,
                           Shishi_asn1 encapreppart,
                           char **t);
```

Extract client time from EncAPRepPart.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encapreppart | EncAPRepPart as allocated by shishi_encapreppart(). | |
| t | newly allocated zero-terminated character array with client time. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_ctime_set ()

```
int
shishi_encapreppart_ctime_set (Shishi *handle,
                               Shishi_asn1 encapreppart,
                               const char *t);
```

Store client time in EncAPRepPart.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encapreppart | EncAPRepPart as allocated by shishi_encapreppart(). | |
| t | string with generalized time value to store in EncAPRepPart. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_cusec_get ()

```
int
shishi_encapreppart_cusec_get (Shishi *handle,
                               Shishi_asn1 encapreppart,
                               uint32_t *cusec);
```

Extract client microseconds field from EncAPRepPart.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encapreppart | EncAPRepPart as allocated by shishi_encapreppart(). | |
| cusec | output integer with client microseconds field. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_cusec_set ()

```
int
shishi_encapreppart_cusec_set (Shishi *handle,
                               Shishi_asn1 encapreppart,
                               uint32_t cusec);
```

Set the cusec field in the Authenticator.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encapreppart | EncAPRepPart as allocated by shishi_encapreppart(). | |
| cusec | client microseconds to set in authenticator, 0-999999. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_print ()

```
int
shishi_encapreppart_print (Shishi *handle,
                           FILE *fh,
                           Shishi_asn1 encapreppart);
```

Print ASCII armored DER encoding of EncAPRepPart to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| encapreppart | EncAPRepPart to print. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_save ()

```
int
shishi_encapreppart_save (Shishi *handle,
                          FILE *fh,
                          Shishi_asn1 encapreppart);
```

Save DER encoding of EncAPRepPart to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| encapreppart | EncAPRepPart to save. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_to_file ()

```
int
shishi_encapreppart_to_file (Shishi *handle,
                             Shishi_asn1 encapreppart,
                             int filetype,
                             const char *filename);
```

Write EncAPRepPart to file in specified TYPE. The file will be truncated if it exists.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encapreppart | EncAPRepPart to save. | |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. | |
| filename | input variable with filename to write to. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_read ()

```
int
shishi_encapreppart_read (Shishi *handle,
                          FILE *fh,
                          Shishi_asn1 *encapreppart);
```

Read DER encoded EncAPRepPart from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| encapreppart | output variable with newly allocated EncAPRepPart. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_parse ()

```
int
shishi_encapreppart_parse (Shishi *handle,
                           FILE *fh,
                           Shishi_asn1 *encapreppart);
```

Read ASCII armored DER encoded EncAPRepPart from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| encapreppart | output variable with newly allocated EncAPRepPart. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_from_file ()

```
int
shishi_encapreppart_from_file (Shishi *handle,
                               Shishi_asn1 *encapreppart,
                               int filetype,
                               const char *filename);
```

Read EncAPRepPart from file in specified TYPE.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encapreppart | output variable with newly allocated EncAPRepPart. |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. |
| filename | input variable with filename to read from. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_get_key ()

```
int
shishi_encapreppart_get_key (Shishi *handle,
                             Shishi_asn1 encapreppart,
                             Shishi_key **key);
```

Extract the subkey from the encrypted AP-REP part.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encapreppart | input EncAPRepPart variable. |
| key | newly allocated key. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_seqnumber_get ()

```
int
shishi_encapreppart_seqnumber_get (Shishi *handle,
                                   Shishi_asn1 encapreppart,
                                   uint32_t *seqnumber);
```

Extract sequence number field from EncAPRepPart.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encapreppart | EncAPRepPart as allocated by shishi_encapreppart(). |
| seqnumber | output integer with sequence number field. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_seqnumber_remove ()

```
int
shishi_encapreppart_seqnumber_remove (Shishi *handle,
                                      Shishi_asn1 encapreppart);
```

Remove sequence number field in EncAPRepPart.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encapreppart | encapreppart as allocated by shishi_encapreppart(). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encapreppart_seqnumber_set ()

```
int
shishi_encapreppart_seqnumber_set (Shishi *handle,
                                   Shishi_asn1 encapreppart,
                                   uint32_t seqnumber);
```

Store sequence number field in EncAPRepPart.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encapreppart | encapreppart as allocated by shishi_encapreppart(). | |
| seqnumber | integer with sequence number field to store in encapreppart. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq ()

```
Shishi_asn1
shishi_apreq (Shishi *handle);
```

This function creates a new AP-REQ, populated with some default values.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the AP-REQ or NULL on failure.

### shishi_apreq_parse ()

```
int
shishi_apreq_parse (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 *apreq);
```

Read ASCII armored DER encoded AP-REQ from file and populate given variable.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| apreq | output variable with newly allocated AP-REQ. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_from_file ()

```
int
shishi_apreq_from_file (Shishi *handle,
                        Shishi_asn1 *apreq,
                        int filetype,
                        const char *filename);
```

Read AP-REQ from file in specified TYPE.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| apreq | output variable with newly allocated AP-REQ. |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. |
| filename | input variable with filename to read from. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_print ()

```
int
shishi_apreq_print (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 apreq);
```

Print ASCII armored DER encoding of AP-REQ to file.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for writing. |
| apreq | AP-REQ to print. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_to_file ()

```
int
shishi_apreq_to_file (Shishi *handle,
                      Shishi_asn1 apreq,
                      int filetype,
                      const char *filename);
```

Write AP-REQ to file in specified TYPE. The file will be truncated if it exists.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| apreq | AP-REQ to save. |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. |
| filename | input variable with filename to write to. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_read ()

```
int
```

```
shishi_apreq_read (Shishi *handle,
                   FILE *fh,
                   Shishi_asn1 *apreq);
```

Read DER encoded AP-REQ from file and populate given variable.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for reading. | |
| apreq | output variable with newly allocated AP-REQ. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_save ()

```
int
shishi_apreq_save (Shishi *handle,
                   FILE *fh,
                   Shishi_asn1 apreq);
```

Save DER encoding of AP-REQ to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| apreq | AP-REQ to save. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_set_ticket ()

```
int
shishi_apreq_set_ticket (Shishi *handle,
                         Shishi_asn1 apreq,
                         Shishi_asn1 ticket);
```

Copy ticket into AP-REQ.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |

| | | |
|---|---|---|
| apreq | AP-REQ to add ticket field to. | |
| ticket | input ticket to copy into AP-REQ ticket field. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_set_authenticator ()

```
int
shishi_apreq_set_authenticator (Shishi *handle,
                                Shishi_asn1 apreq,
                                int32_t etype,
                                uint32_t kvno,
                                const char *buf,
                                size_t buflen);
```

Set the encrypted authenticator field in the AP-REP. The encrypted data is usually created by calling shishi_encrypt() on the DER encoded authenticator. To save time, you may want to use shishi_apreq_add_authenticator() instead, which calculates the encrypted data and calls this function in one step.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| apreq | AP-REQ to add authenticator field to. | |
| etype | encryption type used to encrypt authenticator. | |
| kvno | version of the key used to encrypt authenticator. | |
| buf | input array with encrypted authenticator. | |
| buflen | size of input array with encrypted authenticator. | |

### Returns

Returns SHISHI_OK on success.

### shishi_apreq_add_authenticator ()

```
int
shishi_apreq_add_authenticator (Shishi *handle,
                                Shishi_asn1 apreq,
                                Shishi_key *key,
                                int keyusage,
                                Shishi_asn1 authenticator);
```

Encrypts DER encoded authenticator using key and store it in the AP-REQ.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| apreq | AP-REQ to add authenticator field to. | |
| key | key to to use for encryption. | |
| keyusage | cryptographic key usage value to use in encryption. | |
| authenticator | authenticator as allocated by shishi_authenticator(). | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_options ()

```
int
shishi_apreq_options (Shishi *handle,
                      Shishi_asn1 apreq,
                      uint32_t *flags);
```

Extract the AP-Options from AP-REQ into output integer.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| apreq | AP-REQ to get options from. | |
| flags | Output integer containing options from AP-REQ. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_use_session_key_p ()

```
int
shishi_apreq_use_session_key_p (Shishi *handle,
                                Shishi_asn1 apreq);
```

Return non-0 iff the "Use session key" option is set in the AP-REQ.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| apreq | AP-REQ as allocated by shishi_apreq(). | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_mutual_required_p ()

```
int
shishi_apreq_mutual_required_p (Shishi *handle,
                                Shishi_asn1 apreq);
```

Return non-0 iff the "Mutual required" option is set in the AP-REQ.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| apreq | AP-REQ as allocated by shishi_apreq(). | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_options_set ()

```
int
shishi_apreq_options_set (Shishi *handle,
                          Shishi_asn1 apreq,
                          uint32_t options);
```

Set the AP-Options in AP-REQ to indicate integer.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| apreq | AP-REQ as allocated by shishi_apreq(). | |
| options | Options to set in AP-REQ. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_options_add ()

```
int
shishi_apreq_options_add (Shishi *handle,
                          Shishi_asn1 apreq,
                          uint32_t option);
```

Add the AP-Options in AP-REQ. Options not set in input parameter `option` are preserved in the AP-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| apreq | AP-REQ as allocated by shishi_apreq(). |
| option | Options to add in AP-REQ. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_options_remove ()

```
int
shishi_apreq_options_remove (Shishi *handle,
                             Shishi_asn1 apreq,
                             uint32_t option);
```

Remove the AP-Options from AP-REQ. Options not set in input parameter `option` are preserved in the AP-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| apreq | AP-REQ as allocated by shishi_apreq(). |
| option | Options to remove from AP-REQ. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_get_ticket ()

```
int
shishi_apreq_get_ticket (Shishi *handle,
                         Shishi_asn1 apreq,
                         Shishi_asn1 *ticket);
```

Extract ticket from AP-REQ.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| apreq | AP-REQ variable to get ticket from. |
| ticket | output variable to hold extracted ticket. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_get_authenticator_etype ()

```
int
shishi_apreq_get_authenticator_etype (Shishi *handle,
                                      Shishi_asn1 apreq,
                                      int32_t *etype);
```

Extract AP-REQ.authenticator.etype.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| apreq | AP-REQ variable to get value from. |
| etype | output variable that holds the value. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_apreq_decrypt ()

```
int
shishi_apreq_decrypt (Shishi *handle,
                      Shishi_asn1 apreq,
                      Shishi_key *key,
                      int keyusage,
                      Shishi_asn1 *authenticator);
```

### shishi_aprep ()

```
Shishi_asn1
shishi_aprep (Shishi *handle);
```

This function creates a new AP-REP, populated with some default values.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |

#### Returns

Returns the authenticator or NULL on failure.

### shishi_aprep_print ()

```
int
shishi_aprep_print (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 aprep);
```

Print ASCII armored DER encoding of AP-REP to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| aprep | AP-REP to print. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_aprep_save ()

```
int
shishi_aprep_save (Shishi *handle,
                   FILE *fh,
                   Shishi_asn1 aprep);
```

Save DER encoding of AP-REP to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| aprep | AP-REP to save. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_aprep_to_file ()

```
int
shishi_aprep_to_file (Shishi *handle,
                      Shishi_asn1 aprep,
                      int filetype,
                      const char *filename);
```

Write AP-REP to file in specified TYPE. The file will be truncated if it exists.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| aprep | AP-REP to save. |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. |
| filename | input variable with filename to write to. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_aprep_read ()

```
int
shishi_aprep_read (Shishi *handle,
                   FILE *fh,
                   Shishi_asn1 *aprep);
```

Read DER encoded AP-REP from file and populate given variable.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| aprep | output variable with newly allocated AP-REP. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_aprep_parse ()

```
int
shishi_aprep_parse (Shishi *handle,
                    FILE *fh,
                    Shishi_asn1 *aprep);
```

Read ASCII armored DER encoded AP-REP from file and populate given variable.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for reading. |
| aprep | output variable with newly allocated AP-REP. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_aprep_from_file ()

```
int
shishi_aprep_from_file (Shishi *handle,
                        Shishi_asn1 *aprep,
                        int filetype,
                        const char *filename);
```

Read AP-REP from file in specified TYPE.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| aprep | output variable with newly allocated AP-REP. | |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. | |
| filename | input variable with filename to read from. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_aprep_decrypt ()

```
int
shishi_aprep_decrypt (Shishi *handle,
                      Shishi_asn1 aprep,
                      Shishi_key *key,
                      int keyusage,
                      Shishi_asn1 *encapreppart);
```

### shishi_aprep_verify ()

```
int
shishi_aprep_verify (Shishi *handle,
                     Shishi_asn1 authenticator,
                     Shishi_asn1 encapreppart);
```

### shishi_aprep_enc_part_set ()

```
int
shishi_aprep_enc_part_set (Shishi *handle,
                           Shishi_asn1 aprep,
                           int etype,
                           const char *buf,
                           size_t buflen);
```

### shishi_aprep_enc_part_add ()

```
int
shishi_aprep_enc_part_add (Shishi *handle,
                           Shishi_asn1 aprep,
                           Shishi_asn1 encticketpart,
                           Shishi_asn1 encapreppart);
```

### shishi_aprep_enc_part_make ()

```
int
shishi_aprep_enc_part_make (Shishi *handle,
                            Shishi_asn1 aprep,
                            Shishi_asn1 encapreppart,
                            Shishi_asn1 authenticator,
                            Shishi_asn1 encticketpart);
```

### shishi_aprep_get_enc_part_etype ()

```
int
shishi_aprep_get_enc_part_etype (Shishi *handle,
                                 Shishi_asn1 aprep,
                                 int32_t *etype);
```

Extract AP-REP.enc-part.etype.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| aprep | AP-REP variable to get value from. | |
| etype | output variable that holds the value. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_kdc_sendrecv ()

```
int
shishi_kdc_sendrecv (Shishi *handle,
                     const char *realm,
                     const char *indata,
                     size_t inlen,
                     char **outdata,
                     size_t *outlen);
```

Send packet to KDC for realm and receive response. The code finds KDC addresses from configuration file, then by querying for SRV records for the realm, and finally by using the realm name as a hostname.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| realm | string with realm name. | |
| indata | Packet to send to KDC. | |
| inlen | Length of *indata* . | |
| outdata | Newly allocated string with data returned from KDC. | |
| outlen | Length of *outdata* . | |

#### Returns

SHISHI_OK on success, SHISHI_KDC_TIMEOUT if a timeout was reached, or other errors.

### shishi_kdc_sendrecv_hint ()

```
int
shishi_kdc_sendrecv_hint (Shishi *handle,
                          const char *realm,
                          const char *indata,
                          size_t inlen,
                          char **outdata,
                          size_t *outlen,
                          Shishi_tkts_hint *hint);
```

Send packet to KDC for realm and receive response. The code finds KDC addresses from configuration file, then by querying for SRV records for the realm, and finally by using the realm name as a hostname.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| realm | string with realm name. | |
| indata | Packet to send to KDC. | |
| inlen | Length of *indata* . | |
| outdata | Newly allocated string with data returned from KDC. | |
| outlen | Length of *outdata* . | |
| hint | a Shishi_tkts_hint structure with flags. | |

#### Returns

SHISHI_OK on success, SHISHI_KDC_TIMEOUT if a timeout was reached, or other errors.

### shishi_encticketpart ()

```
Shishi_asn1
shishi_encticketpart (Shishi *handle);
```

### shishi_encticketpart_key_set ()

```
int
shishi_encticketpart_key_set (Shishi *handle,
                              Shishi_asn1 encticketpart,
                              Shishi_key *key);
```

Set the EncTicketPart.key field to key type and value of supplied key.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encticketpart | input EncTicketPart variable. |
| key | key handle with information to store in encticketpart. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_get_key ()

```
int
shishi_encticketpart_get_key (Shishi *handle,
                              Shishi_asn1 encticketpart,
                              Shishi_key **key);
```

Extract the session key in the Ticket.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encticketpart | input EncTicketPart variable. |
| key | newly allocated key. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_crealm ()

```
int
shishi_encticketpart_crealm (Shishi *handle,
                             Shishi_asn1 encticketpart,
                             char **crealm,
                             size_t *crealmlen);
```

### shishi_encticketpart_crealm_set ()

```
int
shishi_encticketpart_crealm_set (Shishi *handle,
                                 Shishi_asn1 encticketpart,
                                const char *realm);
```

Set the realm field in the KDC-REQ.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encticketpart | input EncTicketPart variable. | |
| realm | input array with name of realm. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_client ()

```
int
shishi_encticketpart_client (Shishi *handle,
                             Shishi_asn1 encticketpart,
                             char **client,
                             size_t *clientlen);
```

Represent client principal name in EncTicketPart as zero-terminated string. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `clientlen` does not include the terminating zero.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| encticketpart | EncTicketPart variable to get client name from. | |
| client | pointer to newly allocated zero terminated string containing principal name. May be NULL (to only populate `clientlen` ). | |
| clientlen | pointer to length of `client` on output, excluding terminating zero. May be NULL (to only populate `client` ). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_clientrealm ()

```
int
shishi_encticketpart_clientrealm (Shishi *handle,
                                 Shishi_asn1 encticketpart,
                                 char **client,
                                 size_t *clientlen);
```

Convert cname and realm fields from EncTicketPart to printable principal name format. The string is allocate by this function, and it is the responsibility of the caller to deallocate it. Note that the output length `clientlen` does not include the terminating zero.

#### Parameters

| | | |
|---|---|---|
| handle | Shishi library handle create by shishi_init(). | |
| encticketpart | EncTicketPart variable to get client name and realm from. | |
| client | pointer to newly allocated zero terminated string containing principal name and realm. May be NULL (to only populate `clientlen` ). | |
| clientlen | pointer to length of `client` on output, excluding terminating zero. May be NULL (to only populate `client` ). | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_cname_set ()

```
int
shishi_encticketpart_cname_set (Shishi *handle,
                                Shishi_asn1 encticketpart,
                                Shishi_name_type name_type,
                                const char *principal);
```

Set the client name field in the EncTicketPart.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encticketpart | input EncTicketPart variable. | |
| name_type | type of principal, see Shishi_name_type, usually SHISHI_NT_UNKNOWN. | |
| principal | input array with principal name. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_print ()

```
int
shishi_encticketpart_print (Shishi *handle,
                            FILE *fh,
                            Shishi_asn1 encticketpart);
```

### shishi_encticketpart_flags_set ()

```
int
shishi_encticketpart_flags_set (Shishi *handle,
                                Shishi_asn1 encticketpart,
                                int flags);
```

Set the EncTicketPart.flags to supplied value.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encticketpart | input EncTicketPart variable. | |
| flags | flags to set in encticketpart. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_transited_set ()

```
int
shishi_encticketpart_transited_set (Shishi *handle,
                                    Shishi_asn1 encticketpart,
                                    int32_t trtype,
                                    const char *trdata,
                                    size_t trdatalen);
```

Set the EncTicketPart.transited field to supplied value.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| encticketpart | input EncTicketPart variable. | |
| trtype | transitedencoding type, e.g. SHISHI_TR_DOMAIN_X500_COMPRESS. | |
| trdata | actual transited realm data. | |
| trdatalen | length of actual transited realm data. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_authtime_set ()

```
int
shishi_encticketpart_authtime_set (Shishi *handle,
                                   Shishi_asn1 encticketpart,
                                   const char *authtime);
```

Set the EncTicketPart.authtime to supplied value.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encticketpart | input EncTicketPart variable. |
| authtime | character buffer containing a generalized time string. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_endtime_set ()

```
int
shishi_encticketpart_endtime_set (Shishi *handle,
                                  Shishi_asn1 encticketpart,
                                  const char *endtime);
```

Set the EncTicketPart.endtime to supplied value.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encticketpart | input EncTicketPart variable. |
| endtime | character buffer containing a generalized time string. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encticketpart_authtime ()

```
int
shishi_encticketpart_authtime (Shishi *handle,
                                Shishi_asn1 encticketpart,
                                char *authtime,
                                size_t *authtimelen);
```

### shishi_encticketpart_authctime ()

```
time_t
shishi_encticketpart_authctime (Shishi *handle,
                                 Shishi_asn1 encticketpart);
```

### shishi_safe ()

```
int
shishi_safe (Shishi *handle,
             Shishi_safe **safe);
```

Create a new SAFE exchange.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| safe | pointer to new structure that holds information about SAFE exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_done ()

```
void
shishi_safe_done (Shishi_safe *safe);
```

Deallocate resources associated with SAFE exchange. This should be called by the application when it no longer need to utilize the SAFE exchange handle.

#### Parameters

| | |
|---|---|
| safe | structure that holds information about SAFE exchange |

### shishi_safe_key ()

```
Shishi_key~*
shishi_safe_key (Shishi_safe *safe);
```

Get key structured from SAFE exchange.

#### Parameters

| | |
|---|---|
| safe | structure that holds information about SAFE exchange |

#### Returns

Returns the key used in the SAFE exchange, or NULL if not yet set or an error occured.

### shishi_safe_key_set ()

```
void
shishi_safe_key_set (Shishi_safe *safe,
                     Shishi_key *key);
```

Set the Key in the SAFE exchange.

#### Parameters

| | |
|---|---|
| safe | structure that holds information about SAFE exchange |
| key | key to store in SAFE. |

### shishi_safe_safe ()

```
Shishi_asn1
shishi_safe_safe (Shishi_safe *safe);
```

Get ASN.1 SAFE structured from SAFE exchange.

#### Parameters

| | |
|---|---|
| safe | structure that holds information about SAFE exchange |

#### Returns

Returns the ASN.1 safe in the SAFE exchange, or NULL if not yet set or an error occured.

### shishi_safe_safe_set ()

```
void
shishi_safe_safe_set (Shishi_safe *safe,
                      Shishi_asn1 asn1safe);
```

Set the KRB-SAFE in the SAFE exchange.

#### Parameters

| | | |
|---|---|---|
| safe | structure that holds information about SAFE exchange | |
| asn1safe | KRB-SAFE to store in SAFE exchange. | |

### shishi_safe_safe_der ()

```
int
shishi_safe_safe_der (Shishi_safe *safe,
                      char **out,
                      size_t *outlen);
```

DER encode SAFE structure. Typically shishi_safe_build() is used to build the SAFE structure first. *out* is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| safe | safe as allocated by shishi_safe(). | |
| out | output array with newly allocated DER encoding of SAFE. | |
| outlen | length of output array with DER encoding of SAFE. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_safe_der_set ()

```
int
shishi_safe_safe_der_set (Shishi_safe *safe,
                          char *der,
                          size_t derlen);
```

DER decode KRB-SAFE and set it SAFE exchange. If decoding fails, the KRB-SAFE in the SAFE exchange remains.

#### Parameters

| | | |
|---|---|---|
| safe | safe as allocated by shishi_safe(). | |
| der | input array with DER encoded KRB-SAFE. | |
| derlen | length of input array with DER encoded KRB-SAFE. | |

#### Returns

Returns SHISHI_OK.

### shishi_safe_print ()

```
int
shishi_safe_print (Shishi *handle,
                   FILE *fh,
                   Shishi_asn1 safe);
```

Print ASCII armored DER encoding of SAFE to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| safe | SAFE to print. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_save ()

```
int
shishi_safe_save (Shishi *handle,
                  FILE *fh,
                  Shishi_asn1 safe);
```

Save DER encoding of SAFE to file.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for writing. | |
| safe | SAFE to save. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_to_file ()

```
int
shishi_safe_to_file (Shishi *handle,
                     Shishi_asn1 safe,
                     int filetype,
                     const char *filename);
```

Write SAFE to file in specified TYPE. The file will be truncated if it exists.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| safe | SAFE to save. | |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. | |
| filename | input variable with filename to write to. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_parse ()

```
int
shishi_safe_parse (Shishi *handle,
                   FILE *fh,
                   Shishi_asn1 *safe);
```

Read ASCII armored DER encoded SAFE from file and populate given variable.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for reading. | |
| safe | output variable with newly allocated SAFE. | |

### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_read ()

```
int
shishi_safe_read (Shishi *handle,
                  FILE *fh,
                  Shishi_asn1 *safe);
```

Read DER encoded SAFE from file and populate given variable.

### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for reading. | |
| safe | output variable with newly allocated SAFE. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_from_file ()

```
int
shishi_safe_from_file (Shishi *handle,
                       Shishi_asn1 *safe,
                       int filetype,
                       const char *filename);
```

Read SAFE from file in specified TYPE.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| safe | output variable with newly allocated SAFE. | |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. | |
| filename | input variable with filename to read from. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_cksum ()

```
int
shishi_safe_cksum (Shishi *handle,
                   Shishi_asn1 safe,
                   int32_t *cksumtype,
                   char **cksum,
                   size_t *cksumlen);
```

Read checksum value from KRB-SAFE. `cksum` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| safe | safe as allocated by shishi_safe(). | |
| cksumtype | output checksum type. | |
| cksum | output array with newly allocated checksum data from SAFE. | |
| cksumlen | output size of output checksum data buffer. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_set_cksum ()

```
int
shishi_safe_set_cksum (Shishi *handle,
                       Shishi_asn1 safe,
                       int32_t cksumtype,
                       const char *cksum,
                       size_t cksumlen);
```

Store checksum value in SAFE. A checksum is usually created by calling shishi_checksum() on some application specific data using the key from the ticket that is being used. To save time, you may want to use shishi_safe_build() instead, which calculates the checksum and calls this function in one step.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| safe | safe as allocated by shishi_safe(). | |
| cksumtype | input checksum type to store in SAFE. | |
| cksum | input checksum data to store in SAFE. | |
| cksumlen | size of input checksum data to store in SAFE. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_user_data ()

```
int
shishi_safe_user_data (Shishi *handle,
                       Shishi_asn1 safe,
                       char **userdata,
                       size_t *userdatalen);
```

Read user data value from KRB-SAFE. `userdata` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| safe | safe as allocated by shishi_safe(). | |
| userdata | output array with newly allocated user data from KRB-SAFE. | |

| | | |
|---|---|---|
| userdatalen | output size of output user data buffer. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_set_user_data ()

```
int
shishi_safe_set_user_data (Shishi *handle,
                           Shishi_asn1 safe,
                           const char *userdata,
                           size_t userdatalen);
```

Set the application data in SAFE.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| safe | safe as allocated by shishi_safe(). | |
| userdata | input user application to store in SAFE. | |
| userdatalen | size of input user application to store in SAFE. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_build ()

```
int
shishi_safe_build (Shishi_safe *safe,
                   Shishi_key *key);
```

Build checksum and set it in KRB-SAFE. Note that this follows RFC 1510bis and is incompatible with RFC 1510, although presumably few implementations use the RFC1510 algorithm.

#### Parameters

| | | |
|---|---|---|
| safe | safe as allocated by shishi_safe(). | |
| key | key for session, used to compute checksum. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_safe_verify ()

```
int
shishi_safe_verify (Shishi_safe *safe,
                    Shishi_key *key);
```

Verify checksum in KRB-SAFE. Note that this follows RFC 1510bis and is incompatible with RFC 1510, although presumably few implementations use the RFC1510 algorithm.

#### Parameters

| | |
|---|---|
| safe | safe as allocated by shishi_safe(). |
| key | key for session, used to verify checksum. |

#### Returns

Returns SHISHI_OK iff successful, SHISHI_SAFE_BAD_KEYTYPE if an incompatible key type is used, or SHISHI_SAFE_VERIFY_
if the actual verification failed.

### shishi_priv ()

```
int
shishi_priv (Shishi *handle,
             Shishi_priv **priv);
```

Create a new PRIV exchange.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| priv | pointer to new structure that holds information about PRIV exchange |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_done ()

```
void
shishi_priv_done (Shishi_priv *priv);
```

Deallocate resources associated with PRIV exchange. This should be called by the application when it no longer need to utilize the PRIV exchange handle.

#### Parameters

| | |
|---|---|
| priv | structure that holds information about PRIV exchange |

### shishi_priv_key ()

```
Shishi_key~*
shishi_priv_key (Shishi_priv *priv);
```

Get key from PRIV exchange.

#### Parameters

| | |
|---|---|
| priv | structure that holds information about PRIV exchange |

#### Returns

Returns the key used in the PRIV exchange, or NULL if not yet set or an error occured.

### shishi_priv_key_set ()

```
void
shishi_priv_key_set (Shishi_priv *priv,
                     Shishi_key *key);
```

Set the Key in the PRIV exchange.

#### Parameters

| | |
|---|---|
| priv | structure that holds information about PRIV exchange |
| key | key to store in PRIV. |

### shishi_priv_priv ()

```
Shishi_asn1
shishi_priv_priv (Shishi_priv *priv);
```

Get ASN.1 PRIV structure in PRIV exchange.

#### Parameters

| | |
|---|---|
| priv | structure that holds information about PRIV exchange |

#### Returns

Returns the ASN.1 priv in the PRIV exchange, or NULL if not yet set or an error occured.

### shishi_priv_priv_set ()

```
void
shishi_priv_priv_set (Shishi_priv *priv,
                      Shishi_asn1 asn1priv);
```

Set the KRB-PRIV in the PRIV exchange.

#### Parameters

| | | |
|---|---|---|
| priv | structure that holds information about PRIV exchange | |
| asn1priv | KRB-PRIV to store in PRIV exchange. | |

### shishi_priv_priv_der ()

```
int
shishi_priv_priv_der (Shishi_priv *priv,
                      char **out,
                      size_t *outlen);
```

DER encode PRIV structure. Typically shishi_priv_build() is used to build the PRIV structure first. *out* is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| priv | priv as allocated by shishi_priv(). | |
| out | output array with newly allocated DER encoding of PRIV. | |
| outlen | length of output array with DER encoding of PRIV. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_priv_der_set ()

```
int
shishi_priv_priv_der_set (Shishi_priv *priv,
                          char *der,
                          size_t derlen);
```

DER decode KRB-PRIV and set it PRIV exchange. If decoding fails, the KRB-PRIV in the PRIV exchange remains.

#### Parameters

| | | |
|---|---|---|
| priv | priv as allocated by shishi_priv(). | |
| der | input array with DER encoded KRB-PRIV. | |
| derlen | length of input array with DER encoded KRB-PRIV. | |

#### Returns

Returns SHISHI_OK.

### shishi_priv_encprivpart ()

```
Shishi_asn1
shishi_priv_encprivpart (Shishi_priv *priv);
```

Get ASN.1 EncPrivPart structure from PRIV exchange.

#### Parameters

| | | |
|---|---|---|
| priv | structure that holds information about PRIV exchange | |

#### Returns

Returns the ASN.1 encprivpart in the PRIV exchange, or NULL if not yet set or an error occured.

### shishi_priv_encprivpart_set ()

```
void
shishi_priv_encprivpart_set (Shishi_priv *priv,
                             Shishi_asn1 asn1encprivpart);
```

Set the ENCPRIVPART in the PRIV exchange.

#### Parameters

| | | |
|---|---|---|
| priv | structure that holds information about PRIV exchange | |
| asn1encprivpart | ENCPRIVPART to store in PRIV exchange. | |

### shishi_priv_encprivpart_der ()

```
int
shishi_priv_encprivpart_der (Shishi_priv *priv,
                             char **out,
                             size_t *outlen);
```

DER encode ENCPRIVPART structure. *out* is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | | |
|---|---|---|
| priv | priv as allocated by shishi_priv(). | |
| out | output array with newly allocated DER encoding of ENCPRIVPART. | |
| outlen | length of output array with DER encoding of ENCPRIVPART. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_encprivpart_der_set ()

```
int
shishi_priv_encprivpart_der_set (Shishi_priv *priv,
                                 char *der,
                                 size_t derlen);
```

DER decode ENCPRIVPART and set it PRIV exchange. If decoding fails, the ENCPRIVPART in the PRIV exchange remains.

#### Parameters

| | | |
|---|---|---|
| priv | priv as allocated by shishi_priv(). | |
| der | input array with DER encoded ENCPRIVPART. | |
| derlen | length of input array with DER encoded ENCPRIVPART. | |

#### Returns

Returns SHISHI_OK.

### shishi_priv_print ()

```
int
shishi_priv_print (Shishi *handle,
                   FILE *fh,
                   Shishi_asn1 priv);
```

Print ASCII armored DER encoding of PRIV to file.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for writing. |
| priv | PRIV to print. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_save ()

```
int
shishi_priv_save (Shishi *handle,
                  FILE *fh,
                  Shishi_asn1 priv);
```

Save DER encoding of PRIV to file.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| fh | file handle open for writing. |
| priv | PRIV to save. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_to_file ()

```
int
shishi_priv_to_file (Shishi *handle,
                     Shishi_asn1 priv,
                     int filetype,
                     const char *filename);
```

Write PRIV to file in specified TYPE. The file will be truncated if it exists.

### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| priv | PRIV to save. |
| filetype | input variable specifying type of file to be written, see Shishi_filetype. |
| filename | input variable with filename to write to. |

### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_parse ()

```
int
shishi_priv_parse (Shishi *handle,
                   FILE *fh,
                   Shishi_asn1 *priv);
```

Read ASCII armored DER encoded PRIV from file and populate given variable.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for reading. | |
| priv | output variable with newly allocated PRIV. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_read ()

```
int
shishi_priv_read (Shishi *handle,
                  FILE *fh,
                  Shishi_asn1 *priv);
```

Read DER encoded PRIV from file and populate given variable.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| fh | file handle open for reading. | |
| priv | output variable with newly allocated PRIV. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_from_file ()

```
int
shishi_priv_from_file (Shishi *handle,
                       Shishi_asn1 *priv,
                       int filetype,
                       const char *filename);
```

Read PRIV from file in specified TYPE.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| priv | output variable with newly allocated PRIV. | |
| filetype | input variable specifying type of file to be read, see Shishi_filetype. | |
| filename | input variable with filename to read from. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_enc_part_etype ()

```
int
shishi_priv_enc_part_etype (Shishi *handle,
                            Shishi_asn1 priv,
                            int32_t *etype);
```

Extract PRIV.enc-part.etype.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle as allocated by shishi_init(). | |
| priv | PRIV variable to get value from. | |
| etype | output variable that holds the value. | |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_set_enc_part ()

```
int
shishi_priv_set_enc_part (Shishi *handle,
                          Shishi_asn1 priv,
                          int32_t etype,
                          const char *encpart,
                          size_t encpartlen);
```

Store encrypted data in PRIV. The encrypted data is usually created by calling shishi_encrypt() on some application specific data using the key from the ticket that is being used. To save time, you may want to use shishi_priv_build() instead, which encryptes the data and calls this function in one step.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| priv | priv as allocated by shishi_priv(). |
| etype | input encryption type to store in PRIV. |
| encpart | input encrypted data to store in PRIV. |
| encpartlen | size of input encrypted data to store in PRIV. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encprivpart_user_data ()

```
int
shishi_encprivpart_user_data (Shishi *handle,
                              Shishi_asn1 encprivpart,
                              char **userdata,
                              size_t *userdatalen);
```

Read user data value from KRB-PRIV. `userdata` is allocated by this function, and it is the responsibility of caller to deallocate it.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encprivpart | encprivpart as allocated by shishi_priv(). |
| userdata | output array with newly allocated user data from KRB-PRIV. |
| userdatalen | output size of output user data buffer. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_encprivpart_set_user_data ()

```
int
shishi_encprivpart_set_user_data (Shishi *handle,
                                  Shishi_asn1 encprivpart,
                                  const char *userdata,
                                  size_t userdatalen);
```

Set the application data in PRIV.

#### Parameters

| | |
|---|---|
| handle | shishi handle as allocated by shishi_init(). |
| encprivpart | encprivpart as allocated by shishi_priv(). |
| userdata | input user application to store in PRIV. |
| userdatalen | size of input user application to store in PRIV. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_build ()

```
int
shishi_priv_build (Shishi_priv *priv,
                   Shishi_key *key);
```

Build checksum and set it in KRB-PRIV. Note that this follows RFC 1510bis and is incompatible with RFC 1510, although presumably few implementations use the RFC1510 algorithm.

#### Parameters

| | |
|---|---|
| priv | priv as allocated by shishi_priv(). |
| key | key for session, used to encrypt data. |

#### Returns

Returns SHISHI_OK iff successful.

### shishi_priv_process ()

```
int
shishi_priv_process (Shishi_priv *priv,
                     Shishi_key *key);
```

Decrypt encrypted data in KRB-PRIV and set the EncPrivPart in the PRIV exchange.

#### Parameters

| | |
|---|---|
| priv | priv as allocated by shishi_priv(). |
| key | key to use to decrypt EncPrivPart. |

#### Returns

Returns SHISHI_OK iff successful, SHISHI_PRIV_BAD_KEYTYPE if an incompatible key type is used, or SHISHI_CRYPTO_ERRO
if the actual decryption failed.

### shishi_authorized_p ()

```
int
shishi_authorized_p (Shishi *handle,
                     Shishi_tkt *tkt,
                     const char *authzname);
```

Simplistic authorization of *authzname* against encrypted client principal name inside ticket. For "basic" authentication type, the principal name must coincide with *authzname* . The "k5login" authentication type attempts the MIT/Heimdal method of parsing the file "~/.k5login" for additional equivalence names.

#### Parameters

| | | |
|---|---|---|
| handle | shishi handle allocated by shishi_init(). | |
| tkt | input variable with ticket info. | |
| authzname | authorization name. | |

#### Returns

Returns 1 if *authzname* is authorized for services by the encrypted principal, and 0 otherwise.

### shishi_authorization_parse ()

```
int
shishi_authorization_parse (const char *authorization);
```

Parse authorization type name.

#### Parameters

| | | |
|---|---|---|
| authorization | name of authorization type, "basic" or "k5login". | |

#### Returns

Returns authorization type corresponding to a string.

### shishi_authorize_strcmp ()

```
int
shishi_authorize_strcmp (Shishi *handle,
                         const char *principal,
                         const char *authzname);
```

Authorization of *authzname* against desired *principal* according to "basic" authentication, i.e., testing for identical strings.

#### Parameters

| | |
|---|---|
| handle | shishi handle allocated by shishi_init(). |
| principal | string with desired principal name. |
| authzname | authorization name. |

### Returns

Returns 1 if `authzname` is authorized for services by the encrypted principal, and 0 otherwise.

### shishi_authorize_k5login ()

```
int
shishi_authorize_k5login (Shishi *handle,
                          const char *principal,
                          const char *authzname);
```

Authorization of `authzname` against desired `principal` in accordance with the MIT/Heimdal authorization method.

### Parameters

| | |
|---|---|
| handle | shishi handle allocated by shishi_init(). |
| principal | string with desired principal name and realm. |
| authzname | authorization name. |

### Returns

Returns 1 if `authzname` is authorized for services by `principal` , and returns 0 otherwise.

### shishi_x509ca_default_file_guess ()

```
char~*
shishi_x509ca_default_file_guess (Shishi *handle);
```

Guesses the default X.509 CA certificate filename; it is $HOME/.shishi/client.ca.

### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

### Returns

Returns default X.509 client certificate filename as a string that has to be deallocated with free() by the caller.

### shishi_x509ca_default_file_set ()

```
void
shishi_x509ca_default_file_set (Shishi *handle,
                                const char *x509cafile);
```

Set the default X.509 CA certificate filename used in the library. The certificate is used during TLS connections with the KDC to authenticate the KDC. The string is copied into the library, so you can dispose of the variable immediately after calling this function.

### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| x509cafile | string with new default x509 client certificate file name, or NULL to reset to default. |

### shishi_x509ca_default_file ()

```
const char~*
shishi_x509ca_default_file (Shishi *handle);
```

Get filename for default X.509 CA certificate.

### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

### Returns

Returns the default X.509 CA certificate filename used in the library. The certificate is used during TLS connections with the KDC to authenticate the KDC. The string is not a copy, so don't modify or deallocate it.

### shishi_x509cert_default_file_guess ()

```
char~*
shishi_x509cert_default_file_guess (Shishi *handle);
```

Guesses the default X.509 client certificate filename; it is $HOME/.shishi/client.certs.

### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

### Returns

Returns default X.509 client certificate filename as a string that has to be deallocated with free() by the caller.

### shishi_x509cert_default_file_set ()

```
void
shishi_x509cert_default_file_set (Shishi *handle,
                                  const char *x509certfile);
```

Set the default X.509 client certificate filename used in the library. The certificate is used during TLS connections with the KDC to authenticate the client. The string is copied into the library, so you can dispose of the variable immediately after calling this function.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| x509certfile | string with new default x509 client certificate file name, or NULL to reset to default. |

### shishi_x509cert_default_file ()

```
const char~*
shishi_x509cert_default_file (Shishi *handle);
```

Get filename for default X.509 certificate.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Returns the default X.509 client certificate filename used in the library. The certificate is used during TLS connections with the KDC to authenticate the client. The string is not a copy, so don't modify or deallocate it.

### shishi_x509key_default_file_guess ()

```
char~*
shishi_x509key_default_file_guess (Shishi *handle);
```

Guesses the default X.509 client key filename; it is $HOME/.shishi/client.key.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Returns default X.509 client key filename as a string that has to be deallocated with free() by the caller.

### shishi_x509key_default_file_set ()

```
void
shishi_x509key_default_file_set (Shishi *handle,
                                 const char *x509keyfile);
```

Set the default X.509 client key filename used in the library. The key is used during TLS connections with the KDC to authenticate the client. The string is copied into the library, so you can dispose of the variable immediately after calling this function.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |
| x509keyfile | string with new default x509 client key file name, or NULL to reset to default. |

### shishi_x509key_default_file ()

```
const char~*
shishi_x509key_default_file (Shishi *handle);
```

Get filename for default X.509 key.

#### Parameters

| | |
|---|---|
| handle | Shishi library handle create by shishi_init(). |

#### Returns

Returns the default X.509 client key filename used in the library. The key is used during TLS connections with the KDC to authenticate the client. The string is not a copy, so don't modify or deallocate it.

### shishi_get_date ()

```
time_t
shishi_get_date (const char *p,
                 const time_t *now);
```

### shishi_xalloc_die ()

```
void
shishi_xalloc_die (void);
```

### shishi_resolv ()

```
Shishi_dns
shishi_resolv (const char *zone,
               uint16_t querytype);
```

Queries the DNS resolver for data of type `querytype` about the domain name `zone` . Currently, the types SHISHI_DNS_TXT and SHISHI_DNS_SRV are the only supported kinds.

After its use, the returned list should be deallocated by a call to shishi_resolv_free().

#### Parameters

| | | |
|---|---|---|
| zone | Domain name of authentication zone, e.g. "EXAMPLE.ORG" | |
| querytype | Type of domain data to query for. | |

#### Returns

Returns a linked list of DNS resource records, or NULL if the query failed.

### shishi_resolv_free ()

```
void
shishi_resolv_free (Shishi_dns rrs);
```

Deallocates a list of DNS resource records returned by a call to shishi_resolv().

#### Parameters

| | | |
|---|---|---|
| rrs | List of DNS RRs as returned by shishi_resolv(). | |

## Types and Values

### enum Shishi_rc

#### Members

| | | |
|---|---|---|
| SHISHI_OK | | |
| SHISHI_ASN1_ERROR | | |
| SHISHI_FOPEN_ERROR | | |
| SHISHI_IO_ERROR | | |
| SHISHI_MALLOC_ERROR | | |
| SHISHI_BASE64_ERROR | | |
| SHISHI_REALM_MISMATCH | | |
| SHISHI_CNAME_MISMATCH | | |
| SHISHI_NONCE_MISMATCH | | |
| SHISHI_TGSREP_BAD_KEYTYPE | | |
| SHISHI_KDCREP_BAD_KEYTYPE | | |
| SHISHI_APREP_BAD_KEYTYPE | | |
| SHISHI_APREP_VERIFY_FAILED | | |
| SHISHI_APREQ_BAD_KEYTYPE | | |
| SHISHI_TOO_SMALL_BUFFER | | |
| SHISHI_DERIVEDKEY_TOO_SMALL | | |
| SHISHI_KEY_TOO_LARGE | | |
| SHISHI_CRYPTO_ERROR | | |
| SHISHI_CRYPTO_INTERNAL_ERROR | | |
| SHISHI_SOCKET_ERROR | | |
| SHISHI_BIND_ERROR | | |
| SHISHI_SENDTO_ERROR | | |
| SHISHI_RECVFROM_ERROR | | |
| SHISHI_CLOSE_ERROR | | |
| SHISHI_KDC_TIMEOUT | | |
| SHISHI_KDC_NOT_KNOWN_FOR_REALM | | |

| | | |
|---|---|---|
| SHISHI_TTY_ERROR | | |
| SHISHI_GOT_KRBERROR | | |
| SHISHI_HANDLE_ERROR | | |
| SHISHI_INVALID_TKTS | | |
| SHISHI_TICKET_BAD_KEYTYPE | | |
| SHISHI_INVALID_KEY | | |
| SHISHI_APREQ_DECRYPT_FAILED | | |
| SHISHI_TICKET_DECRYPT_FAILED | | |
| SHISHI_INVALID_TICKET | | |
| SHISHI_OUT_OF_RANGE | | |
| SHISHI_ASN1_NO_ELEMENT | | |
| SHISHI_SAFE_BAD_KEYTYPE | | |
| SHISHI_SAFE_VERIFY_FAILED | | |
| SHISHI_PKCS5_INVALID_PRF | | |
| SHISHI_PKCS5_INVALID_ITERATION_COUNT | | |
| SHISHI_PKCS5_INVALID_DERIVED_KEY_LENGTH | | |
| SHISHI_PKCS5_DERIVED_KEY_TOO_LONG | | |
| SHISHI_INVALID_PRINCIPAL_NAME | | |
| SHISHI_INVALID_ARGUMENT | | |
| SHISHI_ASN1_NO_VALUE | | |
| SHISHI_CONNECT_ERROR | | |
| SHISHI_VERIFY_FAILED | | |
| SHISHI_PRIV_BAD_KEYTYPE | | |
| SHISHI_FILE_ERROR | | |
| SHISHI_ENCAPREPPART_BAD_KEYTYPE | | |
| SHISHI_GETTIMEOFDAY_ERROR | | |
| SHISHI_KEYTAB_ERROR | | |
| SHISHI_CCACHE_ERROR | | |
| SHISHI_LAST_ERROR | | |

#### enum Shishi_name_type

#### Members

| | | |
|---|---|---|
| SHISHI_NT_UNKNOWN | | |
| SHISHI_NT_PRINCIPAL | | |
| SHISHI_NT_SRV_INST | | |
| SHISHI_NT_SRV_HST | | |
| SHISHI_NT_SRV_XHST | | |
| SHISHI_NT_UID | | |
| SHISHI_NT_X500_PRINCIPAL | | |
| SHISHI_NT_SMTP_NAME | | |
| SHISHI_NT_ENTERPRISE | | |

#### enum Shishi_padata_type

#### Members

| | | |
|---|---|---|
| SHISHI_PA_TGS_REQ | | |
| SHISHI_PA_ENC_TIMESTAMP | | |
| SHISHI_PA_PW_SALT | | |
| SHISHI_PA_RESERVED | | |
| SHISHI_PA_ENC_UNIX_TIME | | |
| SHISHI_PA_SANDIA_SECUREID | | |

| | | |
|---|---|---|
| SHISHI_PA_SESAME | | |
| SHISHI_PA_OSF_DCE | | |
| SHISHI_PA_CYBERSAFE_SECUREID | | |
| SHISHI_PA_AFS3_SALT | | |
| SHISHI_PA_ETYPE_INFO | | |
| SHISHI_PA_SAM_CHALLENGE | | |
| SHISHI_PA_SAM_RESPONSE | | |
| SHISHI_PA_PK_AS_REQ | | |
| SHISHI_PA_PK_AS_REP | | |
| SHISHI_PA_ETYPE_INFO2 | | |
| SHISHI_PA_USE_SPECIFIED_KVNO | | |
| SHISHI_PA_SAM_REDIRECT | | |
| SHISHI_PA_GET_FROM_TYPED_DATA | | |
| SHISHI_TD_PADATA | | |
| SHISHI_PA_SAM_ETYPE_INFO | | |
| SHISHI_PA_ALT_PRINC | | |
| SHISHI_PA_SAM_CHALLENGE2 | | |
| SHISHI_PA_SAM_RESPONSE2 | | |
| SHISHI_PA_EXTRA_TGT | | |
| SHISHI_TD_PKINIT_CMS_CERTIFICATES | | |
| SHISHI_TD_KRB_PRINCIPAL | | |
| SHISHI_TD_KRB_REALM | | |
| SHISHI_TD_TRUSTED_CERTIFIERS | | |
| SHISHI_TD_CERTIFICATE_INDEX | | |
| SHISHI_TD_APP_DEFINED_ERROR | | |
| SHISHI_TD_REQ_NONCE | | |
| SHISHI_TD_REQ_SEQ | | |
| SHISHI_PA_PAC_REQUEST | | |

#### enum Shishi_tr_type

#### Members

| | | |
|---|---|---|
| SHISHI_TR_DOMAIN_X500_COMPRESS | | |

#### enum Shishi_apoptions

#### Members

| | | |
|---|---|---|
| SHISHI_APOPTIONS_RESERVED | | |
| SHISHI_APOPTIONS_USE_SESSION_KEY | | |
| SHISHI_APOPTIONS_MUTUAL_REQUIRED | | |

#### enum Shishi_ticketflags

#### Members

| | | |
|---|---|---|
| SHISHI_TICKETFLAGS_RESERVED | | |
| SHISHI_TICKETFLAGS_FORWARDABLE | | |
| SHISHI_TICKETFLAGS_FORWARDED | | |
| SHISHI_TICKETFLAGS_PROXIABLE | | |
| SHISHI_TICKETFLAGS_PROXY | | |
| SHISHI_TICKETFLAGS_MAY_POSTDATE | | |
| SHISHI_TICKETFLAGS_POSTDATED | | |

| | | |
|---|---|---|
| SHISHI_TICKETFLAGS_INVALID | | |
| SHISHI_TICKETFLAGS_RENEWABLE | | |
| SHISHI_TICKETFLAGS_INITIAL | | |
| SHISHI_TICKETFLAGS_PRE_AUTHENT | | |
| SHISHI_TICKETFLAGS_HW_AUTHENT | | |
| SHISHI_TICKETFLAGS_TRANSITED_POLICY_CHECKED | | |
| SHISHI_TICKETFLAGS_OK_AS_DELEGATE | | |

#### enum Shishi_KDCOptions

#### Members

| | | |
|---|---|---|
| SHISHI_KDCOPTIONS_RESERVED | | |
| SHISHI_KDCOPTIONS_FORWARDABLE | | |
| SHISHI_KDCOPTIONS_FORWARDED | | |
| SHISHI_KDCOPTIONS_PROXIABLE | | |
| SHISHI_KDCOPTIONS_PROXY | | |
| SHISHI_KDCOPTIONS_ALLOW_POSTDATE | | |
| SHISHI_KDCOPTIONS_POSTDATED | | |
| SHISHI_KDCOPTIONS_UNUSED7 | | |
| SHISHI_KDCOPTIONS_RENEWABLE | | |
| SHISHI_KDCOPTIONS_UNUSED9 | | |
| SHISHI_KDCOPTIONS_UNUSED10 | | |
| SHISHI_KDCOPTIONS_UNUSED11 | | |

#### enum Shishi_msgtype

#### Members

| | | |
|---|---|---|
| SHISHI_MSGTYPE_AS_REQ | | |
| SHISHI_MSGTYPE_AS_REP | | |
| SHISHI_MSGTYPE_TGS_REQ | | |
| SHISHI_MSGTYPE_TGS_REP | | |
| SHISHI_MSGTYPE_AP_REQ | | |
| SHISHI_MSGTYPE_AP_REP | | |
| SHISHI_MSGTYPE_RESERVED16 | | |
| SHISHI_MSGTYPE_RESERVED17 | | |
| SHISHI_MSGTYPE_SAFE | | |
| SHISHI_MSGTYPE_PRIV | | |
| SHISHI_MSGTYPE_CRED | | |
| SHISHI_MSGTYPE_ERROR | | |

#### enum Shishi_lrtype

#### Members

| | | |
|---|---|---|
| SHISHI_LRTYPE_LAST_INITIAL_TGT_REQUEST | | |
| SHISHI_LRTYPE_LAST_INITIAL_REQUEST | | |
| SHISHI_LRTYPE_NEWEST_TGT_ISSUE | | |
| SHISHI_LRTYPE_LAST_RENEWAL | | |
| SHISHI_LRTYPE_LAST_REQUEST | | |

#### enum Shishi_etype

#### Members

| | | |
|---|---|---|
| SHISHI_NULL | | |
| SHISHI_DES_CBC_CRC | | |
| SHISHI_DES_CBC_MD4 | | |
| SHISHI_DES_CBC_MD5 | | |
| SHISHI_DES_CBC_NONE | | |
| SHISHI_DES3_CBC_NONE | | |
| SHISHI_DES3_CBC_HMAC_SHA1_KD | | |
| SHISHI_AES128_CTS_HMAC_SHA1_96 | | |
| SHISHI_AES256_CTS_HMAC_SHA1_96 | | |
| SHISHI_ARCFOUR_HMAC | | |
| SHISHI_ARCFOUR_HMAC_EXP | | |

#### enum Shishi_cksumtype

#### Members

| | | |
|---|---|---|
| SHISHI_CRC32 | | |
| SHISHI_RSA_MD4 | | |
| SHISHI_RSA_MD4_DES | | |
| SHISHI_DES_MAC | | |
| SHISHI_DES_MAC_K | | |
| SHISHI_RSA_MD4_DES_K | | |
| SHISHI_RSA_MD5 | | |
| SHISHI_RSA_MD5_DES | | |
| SHISHI_RSA_MD5_DES_GSS | | |
| SHISHI_HMAC_SHA1_DES3_KD | | |
| SHISHI_HMAC_SHA1_96_AES128 | | |
| SHISHI_HMAC_SHA1_96_AES256 | | |
| SHISHI_ARCFOUR_HMAC_MD5 | | |
| SHISHI_KRB5_GSSAPI_CKSUM | | |
| SHISHI_NO_CKSUMTYPE | | |

#### enum Shishi_filetype

#### Members

| | | |
|---|---|---|
| SHISHI_FILETYPE_TEXT | | |
| SHISHI_FILETYPE_DER | | |
| SHISHI_FILETYPE_HEX | | |
| SHISHI_FILETYPE_BASE64 | | |
| SHISHI_FILETYPE_BINARY | | |

#### enum Shishi_outputtype

#### Members

| | | |
|---|---|---|
| SHISHI_OUTPUTTYPE_NULL | | |
| SHISHI_OUTPUTTYPE_STDERR | | |
| SHISHI_OUTPUTTYPE_SYSLOG | | |

#### enum Shishi_authorization

#### Members

| | | |
|---|---|---|
| SHISHI_AUTHORIZATION_BASIC | | |
| SHISHI_AUTHORIZATION_K5LOGIN | | |

#### enum Shishi_keyusage

#### Members

| | | |
|---|---|---|
| SHISHI_KEYUSAGE_ASREQ_PA_ENC_TIMESTAMP | | |
| SHISHI_KEYUSAGE_ENCTICKETPART | | |
| SHISHI_KEYUSAGE_ENCASREPPART | | |
| SHISHI_KEYUSAGE_TGSREQ_AUTHORIZATIONDATA_TGS_SESSION_KEY | | |
| SHISHI_KEYUSAGE_TGSREQ_AUTHORIZATIONDATA_TGS_AUTHENTICATOR_KEY | | |
| SHISHI_KEYUSAGE_TGSREQ_APREQ_AUTHENTICATOR_CKSUM | | |
| SHISHI_KEYUSAGE_TGSREQ_APREQ_AUTHENTICATOR | | |
| SHISHI_KEYUSAGE_ENCTGSREPPART_SESSION_KEY | | |
| SHISHI_KEYUSAGE_ENCTGSREPPART_AUTHENTICATOR_KEY | | |
| SHISHI_KEYUSAGE_APREQ_AUTHENTICATOR_CKSUM | | |
| SHISHI_KEYUSAGE_APREQ_AUTHENTICATOR | | |
| SHISHI_KEYUSAGE_ENCAPREPPART | | |
| SHISHI_KEYUSAGE_KRB_PRIV | | |
| SHISHI_KEYUSAGE_KRB_CRED | | |
| SHISHI_KEYUSAGE_KRB_SAFE | | |
| SHISHI_KEYUSAGE_KRB_ERROR | | |
| SHISHI_KEYUSAGE_AD_KDCISSUED | | |
| SHISHI_KEYUSAGE_TICKET_EXTENSION | | |
| SHISHI_KEYUSAGE_TICKET_EXTENSION_AUTHORIZATION | | |
| SHISHI_KEYUSAGE_GSS_R1 | | |
| SHISHI_KEYUSAGE_GSS_R2 | | |
| SHISHI_KEYUSAGE_GSS_R3 | | |
| SHISHI_KEYUSAGE_ACCEPTOR_SEAL | | |
| SHISHI_KEYUSAGE_ACCEPTOR_SIGN | | |
| SHISHI_KEYUSAGE_INITIATOR_SEAL | | |
| SHISHI_KEYUSAGE_INITIATOR_SIGN | | |
| SHISHI_KEYUSAGE_KCMD_DES | | |
| SHISHI_KEYUSAGE_KCMD_INPUT | | |
| SHISHI_KEYUSAGE_KCMD_OUTPUT | | |
| SHISHI_KEYUSAGE_KCMD_STDERR_INPUT | | |
| SHISHI_KEYUSAGE_KCMD_STDERR_OUTPUT | | |

#### enum Shishi_krb_error

#### Members

| | | |
|---|---|---|
| SHISHI_KDC_ERR_NONE | | |
| SHISHI_KDC_ERR_NAME_EXP | | |
| SHISHI_KDC_ERR_SERVICE_EXP | | |
| SHISHI_KDC_ERR_BAD_PVNO | | |
| SHISHI_KDC_ERR_C_OLD_MAST_KVNO | | |
| SHISHI_KDC_ERR_S_OLD_MAST_KVNO | | |
| SHISHI_KDC_ERR_C_PRINCIPAL_UNKNOWN | | |
| SHISHI_KDC_ERR_S_PRINCIPAL_UNKNOWN | | |

| | | |
|---|---|---|
| SHISHI_KDC_ERR_PRINCIPAL_NOT_UNIQUE | | |
| SHISHI_KDC_ERR_NULL_KEY | | |
| SHISHI_KDC_ERR_CANNOT_POSTDATE | | |
| SHISHI_KDC_ERR_NEVER_VALID | | |
| SHISHI_KDC_ERR_POLICY | | |
| SHISHI_KDC_ERR_BADOPTION | | |
| SHISHI_KDC_ERR_ETYPE_NOSUPP | | |
| SHISHI_KDC_ERR_SUMTYPE_NOSUPP | | |
| SHISHI_KDC_ERR_PADATA_TYPE_NOSUPP | | |
| SHISHI_KDC_ERR_TRTYPE_NOSUPP | | |
| SHISHI_KDC_ERR_CLIENT_REVOKED | | |
| SHISHI_KDC_ERR_SERVICE_REVOKED | | |
| SHISHI_KDC_ERR_TGT_REVOKED | | |
| SHISHI_KDC_ERR_CLIENT_NOTYET | | |
| SHISHI_KDC_ERR_SERVICE_NOTYET | | |
| SHISHI_KDC_ERR_KEY_EXPIRED | | |
| SHISHI_KDC_ERR_PREAUTH_FAILED | | |
| SHISHI_KDC_ERR_PREAUTH_REQUIRED | | |
| SHISHI_KDC_ERR_SERVER_NOMATCH | | |
| SHISHI_KDC_ERR_MUST_USE_USER2USER | | |
| SHISHI_KDC_ERR_PATH_NOT_ACCPETED | | |
| SHISHI_KDC_ERR_SVC_UNAVAILABLE | | |
| SHISHI_KRB_AP_ERR_BAD_INTEGRITY | | |
| SHISHI_KRB_AP_ERR_TKT_EXPIRED | | |
| SHISHI_KRB_AP_ERR_TKT_NYV | | |
| SHISHI_KRB_AP_ERR_REPEAT | | |
| SHISHI_KRB_AP_ERR_NOT_US | | |
| SHISHI_KRB_AP_ERR_BADMATCH | | |
| SHISHI_KRB_AP_ERR_SKEW | | |
| SHISHI_KRB_AP_ERR_BADADDR | | |
| SHISHI_KRB_AP_ERR_BADVERSION | | |
| SHISHI_KRB_AP_ERR_MSG_TYPE | | |
| SHISHI_KRB_AP_ERR_MODIFIED | | |
| SHISHI_KRB_AP_ERR_BADORDER | | |
| SHISHI_KRB_AP_ERR_BADKEYVER | | |
| SHISHI_KRB_AP_ERR_NOKEY | | |
| SHISHI_KRB_AP_ERR_MUT_FAIL | | |
| SHISHI_KRB_AP_ERR_BADDIRECTION | | |
| SHISHI_KRB_AP_ERR_METHOD | | |
| SHISHI_KRB_AP_ERR_BADSEQ | | |
| SHISHI_KRB_AP_ERR_INAPP_CKSUM | | |
| SHISHI_KRB_AP_PATH_NOT_ACCEPTED | | |
| SHISHI_KRB_ERR_RESPONSE_TOO_BIG | | |
| SHISHI_KRB_ERR_GENERIC | | |
| SHISHI_KRB_ERR_FIELD_TOOLONG | | |
| SHISHI_KDC_ERROR_CLIENT_NOT_TRUSTED | | |
| SHISHI_KDC_ERROR_KDC_NOT_TRUSTED | | |
| SHISHI_KDC_ERROR_INVALID_SIG | | |
| SHISHI_KDC_ERR_KEY_TOO_WEAK | | |
| SHISHI_KDC_ERR_CERTIFICATE_MISMATCH | | |
| SHISHI_KRB_AP_ERR_NO_TGT | | |
| SHISHI_KDC_ERR_WRONG_REALM | | |
| SHISHI_KRB_AP_ERR_USER_TO_USER_REQUIRED | | |
| SHISHI_KDC_ERR_CANT_VERIFY_CERTIFICATE | | |
| SHISHI_KDC_ERR_INVALID_CERTIFICATE | | |
| SHISHI_KDC_ERR_REVOKED_CERTIFICATE | | |

| | | |
|---|---|---|
| SHISHI_KDC_ERR_REVOCATION_STATUS_UNKNOWN | | |
| SHISHI_KDC_ERR_REVOCATION_STATUS_UNAVAILABLE | | |
| SHISHI_KDC_ERR_CLIENT_NAME_MISMATCH | | |
| SHISHI_KDC_ERR_KDC_NAME_MISMATCH | | |
| SHISHI_LAST_ERROR_CODE | | |

### enum Shishi_tkts_hintflags

#### Members

| | | |
|---|---|---|
| SHISHI_TKTSHINTFLAGS_ACCEPT_EXPIRED | | |
| SHISHI_TKTSHINTFLAGS_NON_INTERACTIVE | | |

### struct Shishi_tkts_hint

```
struct Shishi_tkts_hint {
    int startpos;
    char *server;
    char *serverrealm;
    char *client;
    char *clientrealm;
    int flags;
    Shishi_ticketflags tktflags;
    Shishi_KDCOptions kdcoptions;
    int32_t etype;
    char *passwd;
    time_t starttime;
    time_t endtime;
    time_t renew_till;
    int32_t preauthetype;
    char *preauthsalt;
    size_t preauthsaltlen;
    char *preauths2kparams;
    size_t preauths2kparamslen;
};
```

### struct Shishi_dns_st

```
struct Shishi_dns_st {
    struct Shishi_dns_st *next;

    uint16_t class;
    uint16_t type;
    uint32_t ttl;

    void *rr;
};
```

### struct Shishi_dns_srv_st

```
struct Shishi_dns_srv_st {
    uint16_t priority;
    uint16_t weight;
    uint16_t port;
    char name[256];
};
```

#### SHISHI_DNS_IN

```
# define SHISHI_DNS_IN 1
```

#### SHISHI_DNS_TXT

```
# define SHISHI_DNS_TXT 16
```

#### SHISHI_DNS_SRV

```
# define SHISHI_DNS_SRV 33
```

#### Shishi_dns

```
  typedef struct Shishi_dns_st *Shishi_dns;
```

#### Shishi_dns_srv

```
  typedef struct Shishi_dns_srv_st *Shishi_dns_srv;
```

#### Shishi

```
  typedef struct Shishi Shishi;
```

#### Shishi_tkt

```
  typedef struct Shishi_tkt Shishi_tkt;
```

#### Shishi_tkts

```
  typedef struct Shishi_tkts Shishi_tkts;
```

#### Shishi_as

```
  typedef struct Shishi_as Shishi_as;
```

#### Shishi_tgs

```
  typedef struct Shishi_tgs Shishi_tgs;
```

#### Shishi_ap

```
typedef struct Shishi_ap Shishi_ap;
```

#### Shishi_key

```
typedef struct Shishi_key Shishi_key;
```

#### Shishi_keys

```
typedef struct Shishi_keys Shishi_keys;
```

#### Shishi_safe

```
typedef struct Shishi_safe Shishi_safe;
```

#### Shishi_priv

```
typedef struct Shishi_priv Shishi_priv;
```

#### Shishi_asn1

```
typedef ASN1_TYPE Shishi_asn1;
```

#### Shishi_crypto

```
typedef struct Shishi_crypto Shishi_crypto;
```

#### SHISHI_GENERALIZEDTIME_LENGTH

```
# define SHISHI_GENERALIZEDTIME_LENGTH 15
```

#### SHISHI_GENERALIZEDTIMEZ_LENGTH

```
# define SHISHI_GENERALIZEDTIMEZ_LENGTH (SHISHI_GENERALIZEDTIME_LENGTH + 1)
```

## 1.2 shishi-version.h

shishi-version.h — version symbols

### Types and Values

| | |
|---|---|
| #define | SHISHI_VERSION |
| #define | SHISHI_VERSION_MAJOR |
| #define | SHISHI_VERSION_MINOR |
| #define | SHISHI_VERSION_PATCH |
| #define | SHISHI_VERSION_NUMBER |

## Description

The shishi-version.h file contains version symbols. It should not be included directly, only via shishi.h.

## Functions

## Types and Values

### SHISHI_VERSION

```
# define SHISHI_VERSION "1.0.3"
```

Pre-processor symbol with a string that describe the header file version number. Used together with shishi_check_version() to verify header file and run-time library consistency.

### SHISHI_VERSION_MAJOR

```
# define SHISHI_VERSION_MAJOR 1
```

Pre-processor symbol with a decimal value that describe the major level of the header file version number. For example, when the header version is 1.2.3 this symbol will be 1.

Since: 1.0.3

### SHISHI_VERSION_MINOR

```
# define SHISHI_VERSION_MINOR 0
```

Pre-processor symbol with a decimal value that describe the minor level of the header file version number. For example, when the header version is 1.2.3 this symbol will be 2.

Since: 1.0.3

### SHISHI_VERSION_PATCH

```
# define SHISHI_VERSION_PATCH 3
```

Pre-processor symbol with a decimal value that describe the patch level of the header file version number. For example, when the header version is 1.2.3 this symbol will be 3.

Since: 1.0.3

### SHISHI_VERSION_NUMBER

```
# define SHISHI_VERSION_NUMBER 0x010003
```

Pre-processor symbol with a hexadecimal value describing the header file version number. For example, when the header version is 1.2.3 this symbol will have the value 0x010203.

Since: 1.0.3

# Chapter 2

# Index