

Shishi API Reference Manual

COLLABORATORS

	<i>TITLE :</i> Shishi API Reference Manual		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		May 20, 2010	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Shishi API Reference Manual	1
1.1	shishi	2
2	Index	252

List of Figures

1.1	Source code layout of Shishi	2
-----	--	---

Chapter 1

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).

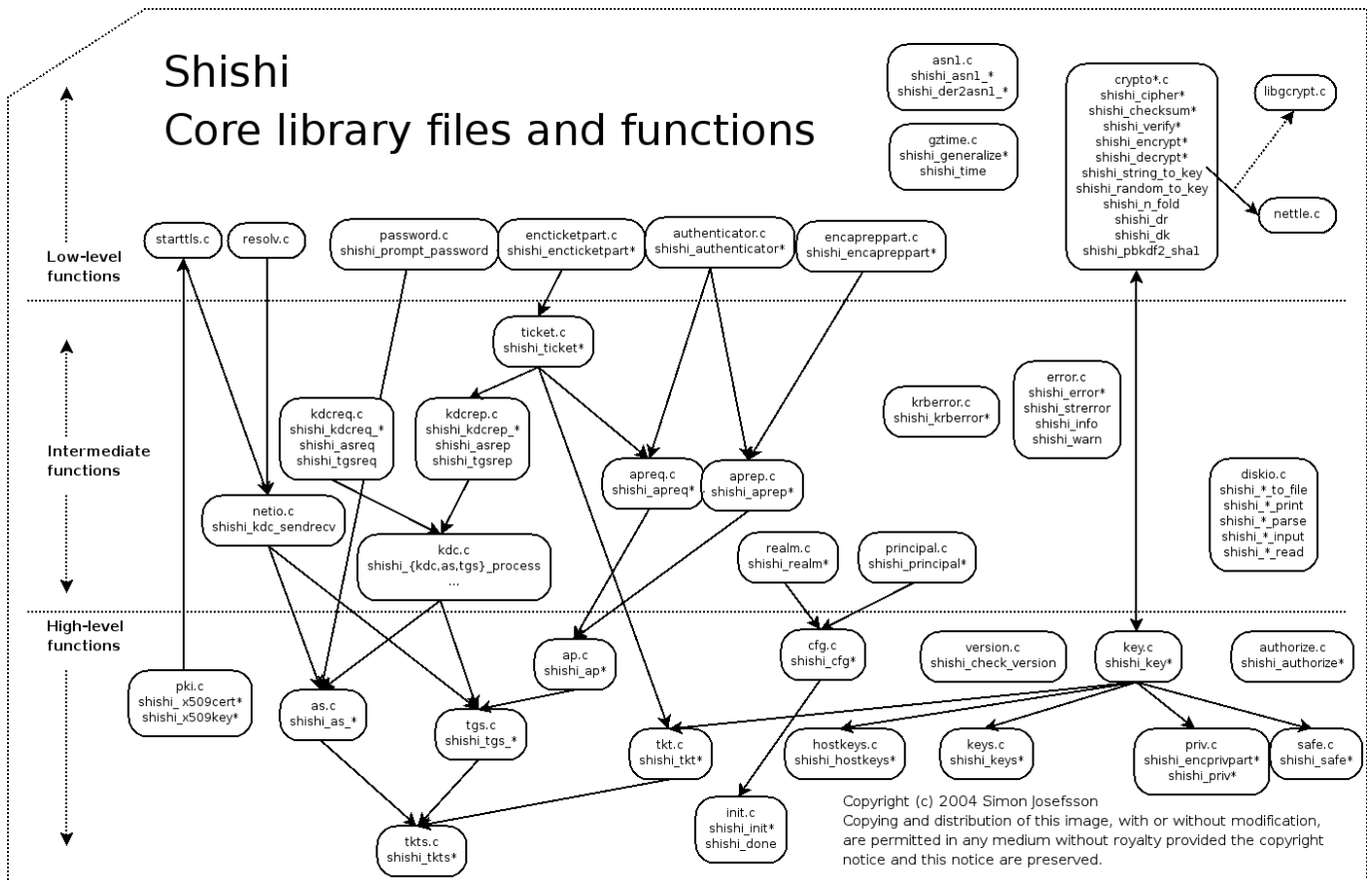


Figure 1.1: Source code layout of Shishi

1.1 shishi

shishi —

Synopsis

```
#define SHISHI_VERSION
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;
typedef             Shishi_tkts_hint;
#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_asnl;
typedef             Shishi_crypto;
#define             SHISHI_GENERALIZEDTIME_LENGTH
#define             SHISHI_GENERALIZEDTIMEZ_LENGTH
void                (*shishi_alloc_fail_function)
Shishi *            shishi
Shishi *            shishi_server
void                shishi_done
int                 shishi_init
int                 shishi_init_with_paths
                                (void);
                                (void);
                                (void);
                                (Shishi *handle);
                                (Shishi **handle);
                                (Shishi **handle,
                                const char *tktsfile,
                                const char *systemcfgfile,
                                const char *usercfgfile);
                                (Shishi **handle);
                                (Shishi **handle,
                                const char *systemcfgfile);
                                (Shishi *handle,
                                const char *option);
                                (Shishi *handle,
                                const char *cfg);
                                (Shishi *handle,
                                FILE *fh);
                                (Shishi *handle);
                                (Shishi *handle);
                                (Shishi *handle);
                                (Shishi *handle,
                                const char *file);
                                (Shishi *handle,
                                int32_t **etypes);
                                (Shishi *handle);
                                (Shishi *handle,
                                char *value);
                                (Shishi *handle,
                                char *value);
                                (int err);
                                (Shishi *handle);
                                (Shishi *handle);
                                (Shishi *handle,
                                const char *errstr);
                                (Shishi *handle,

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

```

		const char *format, ...);
int	shishi_error_outputtype	(Shishi *handle);
void	shishi_error_set_outputtype	(Shishi *handle, int type);
void	shishi_info	(Shishi *handle, const char *format, ...);
void	shishi_warn	(Shishi *handle, const char *format, ...);
void	shishi_verbose	(Shishi *handle, const char *format, ...);
char *	shishi_realm_default_guess	(void);
const char *	shishi_realm_default	(Shishi *handle);
void	shishi_realm_default_set	(Shishi *handle, const char *realm);
char *	shishi_realm_for_server_file	(Shishi *handle, char *server);
char *	shishi_realm_for_server_dns	(Shishi *handle, char *server);
char *	shishi_realm_for_server	(Shishi *handle, char *server);
char *	shishi_principal_default_guess	(void);
const char *	shishi_principal_default	(Shishi *handle);
void	shishi_principal_default_set	(Shishi *handle, const char *principal);
int	shishi_principal_name	(Shishi *handle, Shishi_asn1 namenode, const char *namefield, char **out, size_t *outlen);
int	shishi_principal_name_realm	(Shishi *handle, Shishi_asn1 namenode, const char *namefield, Shishi_asn1 realmnode, const char *realmfield, char **out, size_t *outlen);
int	shishi_principal_name_set	(Shishi *handle, Shishi_asn1 namenode, const char *namefield, Shishi_name_type name_type, const char *name[]);
int	shishi_principal_set	(Shishi *handle, Shishi_asn1 namenode, const char *namefield, const char *name);
int	shishi_parse_name	(Shishi *handle, const char *name, char **principal, char **realm);
int	shishi_derive_default_salt	(Shishi *handle, const char *name, char **salt);
char *	shishi_server_for_local_service	(Shishi *handle,

Shishi_asn1	shishi_ticket	const char *service);
int	shishi_ticket_server	(Shishi *handle);
		(Shishi *handle,
		Shishi_asn1 ticket,
		char **server,
		size_t *serverlen);
int	shishi_ticket_sname_set	(Shishi *handle,
		Shishi_asn1 ticket,
		Shishi_name_type name_type,
		char *sname[]);
int	shishi_ticket_srealmserver_set	(Shishi *handle,
		Shishi_asn1 ticket,
		const char *realm,
		const char *server);
int	shishi_ticket_set_server	(Shishi *handle,
		Shishi_asn1 ticket,
		const char *server);
int	shishi_ticket_realm_get	(Shishi *handle,
		Shishi_asn1 ticket,
		char **realm,
		size_t *realmrlen);
int	shishi_ticket_realm_set	(Shishi *handle,
		Shishi_asn1 ticket,
		const char *realm);
int	shishi_ticket_get_enc_part_etype	(Shishi *handle,
		Shishi_asn1 ticket,
		int32_t *etype);
int	shishi_ticket_set_enc_part	(Shishi *handle,
		Shishi_asn1 ticket,
		int32_t etype,
		uint32_t kvno,
		const char *buf,
		size_t buflen);
int	shishi_ticket_add_enc_part	(Shishi *handle,
		Shishi_asn1 ticket,
		Shishi_key *key,
		Shishi_asn1 encticketpart);
int	shishi_ticket_decrypt	(Shishi *handle,
		Shishi_asn1 ticket,
		Shishi_key *key,
		Shishi_asn1 *encticketpart);
Shishi_asn1	shishi_tkt_ticket	(Shishi_tkt *tkt);
void	shishi_tkt_ticket_set	(Shishi_tkt *tkt,
		Shishi_asn1 ticket);
Shishi_asn1	shishi_tkt_kdcrep	(Shishi_tkt *tkt);
Shishi_asn1	shishi_tkt_enckdcreppart	(Shishi_tkt *tkt);
void	shishi_tkt_enckdcreppart_set	(Shishi_tkt *tkt,
		Shishi_asn1 enckdcreppart);
Shishi_asn1	shishi_tkt_encticketpart	(Shishi_tkt *tkt);
void	shishi_tkt_encticketpart_set	(Shishi_tkt *tkt,
		Shishi_asn1 encticketpart);
Shishi_key *	shishi_tkt_key	(Shishi_tkt *tkt);
int	shishi_tkt_key_set	(Shishi_tkt *tkt,
		Shishi_key *key);
int	shishi_tkt	(Shishi *handle,
		Shishi_tkt **tkt);
Shishi_tkt *	shishi_tkt2	(Shishi *handle,

		Shishi_asn1 ticket, Shishi_asn1 enckdcreppart, Shishi_asn1 kdcrep);
void	shishi_tkt_pretty_print	(Shishi_tkt *tkt, FILE *fh);
int	shishi_tkt_realm	(Shishi_tkt *tkt, char **realm, size_t *realmrlen);
int	shishi_tkt_client	(Shishi_tkt *tkt, char **client, size_t *clientrlen);
int	shishi_tkt_client_p	(Shishi_tkt *tkt, const char *client);
int	shishi_tkt_clientrealm	(Shishi_tkt *tkt, char **client, size_t *clientrlen);
int	shishi_tkt_clientrealm_p	(Shishi_tkt *tkt, const char *client);
int	shishi_tkt_clientrealm_set	(Shishi_tkt *tkt, const char *realm, const char *client);
int	shishi_tkt_serverrealm_set	(Shishi_tkt *tkt, const char *realm, const char *server);
int	shishi_tkt_build	(Shishi_tkt *tkt, Shishi_key *key);
int	shishi_tkt_lastreq	(Shishi_tkt *tkt, char **lrtime, size_t *lrrimelen, int32_t lrtype);
time_t	shishi_tkt_lastreqc	(Shishi_tkt *tkt, Shishi_lrtype lrtype);
void	shishi_tkt_lastreq_pretty_print	(Shishi_tkt *tkt, FILE *fh);
int	shishi_tkt_authtime	(Shishi_tkt *tkt, char **authtime, size_t *authtimelen);
time_t	shishi_tkt_authctime	(Shishi_tkt *tkt);
int	shishi_tkt_starttime	(Shishi_tkt *tkt, char **starttime, size_t *starttimelen);
time_t	shishi_tkt_startctime	(Shishi_tkt *tkt);
int	shishi_tkt_endtime	(Shishi_tkt *tkt, char **endtime, size_t *endtimelen);
time_t	shishi_tkt_endctime	(Shishi_tkt *tkt);
int	shishi_tkt_renew_till	(Shishi_tkt *tkt, char **renewtilltime, size_t *renewtilllen);
time_t	shishi_tkt_renew_tillc	(Shishi_tkt *tkt);
int	shishi_tkt_keytype	(Shishi_tkt *tkt, int32_t *etype);
int32_t	shishi_tkt_keytype_fast	(Shishi_tkt *tkt);
int	shishi_tkt_keytype_p	(Shishi_tkt *tkt, int32_t etype);
int	shishi_tkt_server	(Shishi_tkt *tkt, char **server,

		size_t *serverlen);
int	shishi_tkt_server_p	(Shishi_tkt *tkt, const char *server);
int	shishi_tkt_valid_at_time_p	(Shishi_tkt *tkt, time_t now);
int	shishi_tkt_valid_now_p	(Shishi_tkt *tkt);
int	shishi_tkt_expired_p	(Shishi_tkt *tkt);
int	shishi_tkt_decrypt	(Shishi_tkt *tkt, Shishi_key *key);
void	shishi_tkt_done	(Shishi_tkt *tkt);
int	shishi_tkt_flags	(Shishi_tkt *tkt, uint32_t *flags);
int	shishi_tkt_flags_set	(Shishi_tkt *tkt, uint32_t flags);
int	shishi_tkt_flags_add	(Shishi_tkt *tkt, uint32_t flag);
int	shishi_tkt_forwardable_p	(Shishi_tkt *tkt);
int	shishi_tkt_forwarded_p	(Shishi_tkt *tkt);
int	shishi_tkt_proxiabile_p	(Shishi_tkt *tkt);
int	shishi_tkt_proxy_p	(Shishi_tkt *tkt);
int	shishi_tkt_may_postdate_p	(Shishi_tkt *tkt);
int	shishi_tkt_postdated_p	(Shishi_tkt *tkt);
int	shishi_tkt_invalid_p	(Shishi_tkt *tkt);
int	shishi_tkt_renewable_p	(Shishi_tkt *tkt);
int	shishi_tkt_initial_p	(Shishi_tkt *tkt);
int	shishi_tkt_pre_authent_p	(Shishi_tkt *tkt);
int	shishi_tkt_hw_authent_p	(Shishi_tkt *tkt);
int	shishi_tkt_transited_policy_checked_p	(Shishi_tkt *tkt);
int	shishi_tkt_ok_as_delegate_p	(Shishi_tkt *tkt);
char *	shishi_tkts_default_file_guess	(Shishi *handle);
const char *	shishi_tkts_default_file	(Shishi *handle);
void	shishi_tkts_default_file_set	(Shishi *handle, const char *tktsfile);
Shishi_tkts *	shishi_tkts_default	(Shishi *handle);
int	shishi_tkts_default_to_file	(Shishi_tkts *tkts);
int	shishi_tkts	(Shishi *handle, Shishi_tkts **tkts);
Shishi_tkt *	shishi_tkts_nth	(Shishi_tkts *tkts, int ticketno);
int	shishi_tkts_size	(Shishi_tkts *tkts);
int	shishi_tkts_add	(Shishi_tkts *tkts, Shishi_tkt *tkt);
int	shishi_tkts_new	(Shishi_tkts *tkts, Shishi_asn1 ticket, Shishi_asn1 enckdcreppart, Shishi_asn1 kdcrep);
int	shishi_tkts_remove	(Shishi_tkts *tkts, int ticketno);
int	shishi_tkts_expire	(Shishi_tkts *tkts);
int	shishi_tkts_print_for_service	(Shishi_tkts *tkts, FILE *fh, const char *service);
int	shishi_tkts_print	(Shishi_tkts *tkts, FILE *fh);
int	shishi_tkts_write	(Shishi_tkts *tkts, FILE *fh);

int	shishi_tkts_to_file	(Shishi_tkts *tkts, const char *filename);
int	shishi_tkts_read	(Shishi_tkts *tkts, FILE *fh);
int	shishi_tkts_from_file	(Shishi_tkts *tkts, const char *filename);
void	shishi_tkts_done	(Shishi_tkts **tkts);
int	shishi_tkt_match_p	(Shishi_tkt *tkt, Shishi_tkts_hint *hint);
Shishi_tkt *	shishi_tkts_find	(Shishi_tkts *tkts, Shishi_tkts_hint *hint);
Shishi_tkt *	shishi_tkts_find_for_clientserver	(Shishi_tkts *tkts, const char *client, const char *server);
Shishi_tkt *	shishi_tkts_find_for_server	(Shishi_tkts *tkts, const char *server);
Shishi_tkt *	shishi_tkts_get	(Shishi_tkts *tkts, Shishi_tkts_hint *hint);
Shishi_tkt *	shishi_tkts_get_tgt	(Shishi_tkts *tkts, Shishi_tkts_hint *hint);
Shishi_tkt *	shishi_tkts_get_tgs	(Shishi_tkts *tkts, Shishi_tkts_hint *hint, Shishi_tkt *tgt);
Shishi_tkt *	shishi_tkts_get_for_clientserver	(Shishi_tkts *tkts, const char *client, const char *server);
Shishi_tkt *	shishi_tkts_get_for_server	(Shishi_tkts *tkts, const char *server);
Shishi_tkt *	shishi_tkts_get_for_localservicepasswd	(Shishi_tkts *tkts, const char *service, const char *passwd);
char *	shishi_tkts_default_ccache_guess	(Shishi *handle);
const char *	shishi_tkts_default_ccache	(Shishi *handle);
void	shishi_tkts_default_ccache_set	(Shishi *handle, const char *ccache);
int	shishi_tkts_add_ccache_mem	(Shishi *handle, const char *data, size_t len, Shishi_tkts *tkts);
int	shishi_tkts_add_ccache_file	(Shishi *handle, const char *filename, Shishi_tkts *tkts);
int	shishi_tkts_from_ccache_mem	(Shishi *handle, const char *data, size_t len, Shishi_tkts **outtkts);
int	shishi_tkts_from_ccache_file	(Shishi *handle, const char *filename, Shishi_tkts **outtkts);
int	shishi_enckdcreppart_print	(Shishi *handle, FILE *fh, Shishi_asn1 enckdcreppart);
int	shishi_enckdcreppart_save	(Shishi *handle, FILE *fh, Shishi_asn1 enckdcreppart);
int	shishi_enckdcreppart_parse	(Shishi *handle,

		FILE *fh, Shishi_asn1 *enckdcreppart);
int	shishi_enckdcreppart_read	(Shishi *handle, FILE *fh, Shishi_asn1 *enckdcreppart);
int	shishi_ticket_save	(Shishi *handle, FILE *fh, Shishi_asn1 ticket);
int	shishi_ticket_print	(Shishi *handle, FILE *fh, Shishi_asn1 ticket);
int	shishi_kdc_print	(Shishi *handle, FILE *fh, Shishi_asn1 asreq, Shishi_asn1 asrep, Shishi_asn1 encasreppart);
int	shishi_ticket_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *ticket);
int	shishi_ticket_read	(Shishi *handle, FILE *fh, Shishi_asn1 *ticket);
int	shishi_etype_info_print	(Shishi *handle, FILE *fh, Shishi_asn1 etypeinfo);
int	shishi_etype_info2_print	(Shishi *handle, FILE *fh, Shishi_asn1 etypeinfo2);
int	shishi_padata_print	(Shishi *handle, FILE *fh, Shishi_asn1 padata);
int	shishi_methoddata_print	(Shishi *handle, FILE *fh, Shishi_asn1 methoddata);
Shishi_asn1	shishi_authenticator	(Shishi *handle);
int	shishi_authenticator_set_crealm	(Shishi *handle, Shishi_asn1 authenticator, const char *crealm);
int	shishi_authenticator_set_cname	(Shishi *handle, Shishi_asn1 authenticator, Shishi_name_type name_type, const char *cname[]);
int	shishi_authenticator_client_set	(Shishi *handle, Shishi_asn1 authenticator, const char *client);
int	shishi_authenticator_ctime	(Shishi *handle, Shishi_asn1 authenticator, char **t);
int	shishi_authenticator_ctime_set	(Shishi *handle, Shishi_asn1 authenticator, const char *t);
int	shishi_authenticator_cusec_get	(Shishi *handle, Shishi_asn1 authenticator, uint32_t *cusec);
int	shishi_authenticator_cusec_set	(Shishi *handle, Shishi_asn1 authenticator, uint32_t cusec);

int	shishi_authenticator_seqnumber_get	(Shishi *handle, Shishi_asn1 authenticator, uint32_t *seqnumber);
int	shishi_authenticator_seqnumber_remove	(Shishi *handle, Shishi_asn1 authenticator);
int	shishi_authenticator_seqnumber_set	(Shishi *handle, Shishi_asn1 authenticator, uint32_t seqnumber);
int	shishi_authenticator_client	(Shishi *handle, Shishi_asn1 authenticator, char **client, size_t *clientlen);
int	shishi_authenticator_clientrealm	(Shishi *handle, Shishi_asn1 authenticator, char **client, size_t *clientlen);
int	shishi_authenticator_remove_cksum	(Shishi *handle, Shishi_asn1 authenticator);
int	shishi_authenticator_cksum	(Shishi *handle, Shishi_asn1 authenticator, int32_t *cksumtype, char **cksum, size_t *cksumlen);
int	shishi_authenticator_set_cksum	(Shishi *handle, Shishi_asn1 authenticator, int cksumtype, char *cksum, size_t cksumlen);
int	shishi_authenticator_add_cksum	(Shishi *handle, Shishi_asn1 authenticator, Shishi_key *key, int keyusage, char *data, size_t datalen);
int	shishi_authenticator_add_cksum_type	(Shishi *handle, Shishi_asn1 authenticator, Shishi_key *key, int keyusage, int cksumtype, char *data, size_t datalen);
int	shishi_authenticator_remove_subkey	(Shishi *handle, Shishi_asn1 authenticator);
Shishi_asn1	shishi_authenticator_subkey	(Shishi *handle);
int	shishi_authenticator_get_subkey	(Shishi *handle, Shishi_asn1 authenticator, Shishi_key **subkey);
int	shishi_authenticator_set_subkey	(Shishi *handle, Shishi_asn1 authenticator, int32_t subkeytype, const char *subkey, size_t subkeylen);
int	shishi_authenticator_add_random_subkey	(Shishi *handle, Shishi_asn1 authenticator);
int	shishi_authenticator_add_random_subkey_etype	

		(Shishi *handle, Shishi_asn1 authenticator, int etype);
int	shishi_authenticator_add_subkey	(Shishi *handle, Shishi_asn1 authenticator, Shishi_key *subkey);
int	shishi_authenticator_clear_authorizationdata	(Shishi *handle, Shishi_asn1 authenticator);
int	shishi_authenticator_add_authorizationdata	(Shishi *handle, Shishi_asn1 authenticator, int32_t adtype, const char *addata, size_t addatalen);
int	shishi_authenticator_authorizationdata	(Shishi *handle, Shishi_asn1 authenticator, int32_t *adtype, char **addata, size_t *addatalen, size_t nth);
int	shishi_authenticator_read	(Shishi *handle, FILE *fh, Shishi_asn1 *authenticator);
int	shishi_authenticator_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *authenticator);
int	shishi_authenticator_from_file	(Shishi *handle, Shishi_asn1 *authenticator, int filetype, const char *filename);
int	shishi_authenticator_print	(Shishi *handle, FILE *fh, Shishi_asn1 authenticator);
int	shishi_authenticator_to_file	(Shishi *handle, Shishi_asn1 authenticator, int filetype, const char *filename);
int	shishi_authenticator_save	(Shishi *handle, FILE *fh, Shishi_asn1 authenticator);
int	shishi_as	(Shishi *handle, Shishi_as **as);
void	shishi_as_done	(Shishi_as *as);
Shishi_asn1	shishi_as_req	(Shishi_as *as);
int	shishi_as_req_build	(Shishi_as *as);
void	shishi_as_req_set	(Shishi_as *as, Shishi_asn1 asreq);
int	shishi_as_req_der	(Shishi_as *as, char **out, size_t *outlen);
int	shishi_as_req_der_set	(Shishi_as *as, char *der, size_t derlen);
Shishi_asn1	shishi_as_rep	(Shishi_as *as);
void	shishi_as_rep_set	(Shishi_as *as,

		Shishi_asn1 asrep);
int	shishi_as_rep_build	(Shishi_as *as,
		Shishi_key *key);
int	shishi_as_rep_der	(Shishi_as *as,
		char **out,
		size_t *outlen);
int	shishi_as_rep_der_set	(Shishi_as *as,
		char *der,
		size_t derlen);
Shishi_asn1	shishi_as_krberror	(Shishi_as *as);
int	shishi_as_krberror_der	(Shishi_as *as,
		char **out,
		size_t *outlen);
void	shishi_as_krberror_set	(Shishi_as *as,
		Shishi_asn1 krberror);
Shishi_tkt *	shishi_as_tkt	(Shishi_as *as);
void	shishi_as_tkt_set	(Shishi_as *as,
		Shishi_tkt *tkt);
int	shishi_as_sendrecv	(Shishi_as *as);
int	shishi_as_sendrecv_hint	(Shishi_as *as,
		Shishi_tkts_hint *hint);
int	shishi_as_rep_process	(Shishi_as *as,
		Shishi_key *key,
		const char *password);
int	shishi_tgs	(Shishi *handle,
		Shishi_tgs **tgs);
void	shishi_tgs_done	(Shishi_tgs *tgs);
Shishi_tkt *	shishi_tgs_tgtkt	(Shishi_tgs *tgs);
void	shishi_tgs_tgtkt_set	(Shishi_tgs *tgs,
		Shishi_tkt *tgtkt);
Shishi_ap *	shishi_tgs_ap	(Shishi_tgs *tgs);
Shishi_asn1	shishi_tgs_req	(Shishi_tgs *tgs);
int	shishi_tgs_req_der	(Shishi_tgs *tgs,
		char **out,
		size_t *outlen);
int	shishi_tgs_req_der_set	(Shishi_tgs *tgs,
		char *der,
		size_t derlen);
void	shishi_tgs_req_set	(Shishi_tgs *tgs,
		Shishi_asn1 tgsreq);
int	shishi_tgs_req_build	(Shishi_tgs *tgs);
int	shishi_tgs_req_process	(Shishi_tgs *tgs);
Shishi_asn1	shishi_tgs_rep	(Shishi_tgs *tgs);
int	shishi_tgs_rep_der	(Shishi_tgs *tgs,
		char **out,
		size_t *outlen);
int	shishi_tgs_rep_build	(Shishi_tgs *tgs,
		int keyusage,
		Shishi_key *key);
int	shishi_tgs_rep_process	(Shishi_tgs *tgs);
Shishi_asn1	shishi_tgs_krberror	(Shishi_tgs *tgs);
int	shishi_tgs_krberror_der	(Shishi_tgs *tgs,
		char **out,
		size_t *outlen);
void	shishi_tgs_krberror_set	(Shishi_tgs *tgs,
		Shishi_asn1 krberror);
Shishi_tkt *	shishi_tgs_tkt	(Shishi_tgs *tgs);

void	shishi_tgs_tkt_set	(Shishi_tgs *tgs, Shishi_tkt *tkt);
int	shishi_tgs_sendrecv	(Shishi_tgs *tgs);
int	shishi_tgs_sendrecv_hint	(Shishi_tgs *tgs, Shishi_tkts_hint *hint);
int	shishi_tgs_set_server	(Shishi_tgs *tgs, const char *server);
int	shishi_tgs_set_realm	(Shishi_tgs *tgs, const char *realm);
int	shishi_tgs_set_realmserver	(Shishi_tgs *tgs, const char *realm, const char *server);
int	shishi_kdcreq	(Shishi *handle, char *realm, char *service, Shishi_asn1 *req);
Shishi_asn1	shishi_asreq	(Shishi *handle);
Shishi_asn1	shishi_asreq_rsc	(Shishi *handle, char *realm, char *server, char *client);
Shishi_asn1	shishi_tgsreq	(Shishi *handle);
Shishi_asn1	shishi_tgsreq_rst	(Shishi *handle, char *realm, char *server, Shishi_tkt *tkt);
int	shishi_kdcreq_save	(Shishi *handle, FILE *fh, Shishi_asn1 kdcreq);
int	shishi_kdcreq_print	(Shishi *handle, FILE *fh, Shishi_asn1 kdcreq);
int	shishi_kdcreq_to_file	(Shishi *handle, Shishi_asn1 kdcreq, int filetype, const char *filename);
int	shishi_kdcreq_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *kdcreq);
int	shishi_kdcreq_read	(Shishi *handle, FILE *fh, Shishi_asn1 *kdcreq);
int	shishi_kdcreq_from_file	(Shishi *handle, Shishi_asn1 *kdcreq, int filetype, const char *filename);
int	shishi_asreq_clientrealm	(Shishi *handle, Shishi_asn1 asreq, char **client, size_t *clientlen);
int	shishi_kdcreq_nonce	(Shishi *handle, Shishi_asn1 kdcreq, uint32_t *nonce);
int	shishi_kdcreq_nonce_set	(Shishi *handle, Shishi_asn1 kdcreq, uint32_t nonce);
int	shishi_kdcreq_client	(Shishi *handle,

		Shishi_asn1 kdcreq, char **client, size_t *clientlen);
int	shishi_kdcreq_set_cname	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_name_type name_type, const char *principal);
int	shishi_kdcreq_server	(Shishi *handle, Shishi_asn1 kdcreq, char **server, size_t *serverlen);
int	shishi_kdcreq_set_sname	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_name_type name_type, const char *sname[]);
int	shishi_kdcreq_realm	(Shishi *handle, Shishi_asn1 kdcreq, char **realm, size_t *realmmlen);
int	shishi_kdcreq_realm_get	(Shishi *handle, Shishi_asn1 kdcreq, char **realm, size_t *realmmlen);
int	shishi_kdcreq_set_realm	(Shishi *handle, Shishi_asn1 kdcreq, const char *realm);
int	shishi_kdcreq_set_server	(Shishi *handle, Shishi_asn1 req, const char *service);
int	shishi_kdcreq_set_realmserver	(Shishi *handle, Shishi_asn1 req, char *realm, char *service);
int	shishi_kdcreq_till	(Shishi *handle, Shishi_asn1 kdcreq, char **till, size_t *tillllen);
time_t	shishi_kdcreq_tillc	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_etype	(Shishi *handle, Shishi_asn1 kdcreq, int32_t *etype, int netype);
int	shishi_kdcreq_set_etype	(Shishi *handle, Shishi_asn1 kdcreq, int32_t *etype, int netype);
int	shishi_kdcreq_options	(Shishi *handle, Shishi_asn1 kdcreq, uint32_t *flags);
int	shishi_kdcreq_forwardable_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_forwarded_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_proxiabile_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_proxy_p	(Shishi *handle,

		Shishi_asn1 kdcreq);
int	shishi_kdcreq_allow_postdate_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_postdated_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_renewable_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_disable_transited_check_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_renewable_ok_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_enc_tkt_in_skey_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_renew_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_validate_p	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_options_set	(Shishi *handle, Shishi_asn1 kdcreq, uint32_t options);
int	shishi_kdcreq_options_add	(Shishi *handle, Shishi_asn1 kdcreq, uint32_t option);
int	shishi_kdcreq_clear_padata	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_kdcreq_get_padata	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_padata_type padatatype, char **out, size_t *outlen);
int	shishi_kdcreq_get_padata_tgs	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_asn1 *apreq);
int	shishi_kdcreq_add_padata	(Shishi *handle, Shishi_asn1 kdcreq, int padatatype, const char *data, size_t datalen);
int	shishi_kdcreq_add_padata_tgs	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_asn1 areq);
int	shishi_kdcreq_add_padata_preauth	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_key *key);
int	shishi_kdcreq_build	(Shishi *handle, Shishi_asn1 kdcreq);
int	shishi_as_derive_salt	(Shishi *handle, Shishi_asn1 asreq, Shishi_asn1 asrep, char **salt, size_t *saltlen);
int	shishi_tgs_process	(Shishi *handle, Shishi_asn1 tgsreq, Shishi_asn1 tgsrep, Shishi_asn1 authenticator, Shishi_asn1 oldenckdcreppart,

int	shishi_as_process	Shishi_asn1 *enckdcreppart); (Shishi *handle, Shishi_asn1 asreq, Shishi_asn1 asrep, const char *string, Shishi_asn1 *enckdcreppart);
int	shishi_kdc_process	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_asn1 kdcrep, Shishi_key *key, int keyusage, Shishi_asn1 *enckdcreppart);
int	shishi_kdcreq_sendrecv	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_asn1 *kdcrep);
int	shishi_kdcreq_sendrecv_hint	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_asn1 *kdcrep, Shishi_tkts_hint *hint);
int	shishi_kdc_copy_crealm	(Shishi *handle, Shishi_asn1 kdcrep, Shishi_asn1 encticketpart);
int	shishi_as_check_crealm	(Shishi *handle, Shishi_asn1 asreq, Shishi_asn1 asrep);
int	shishi_kdc_copy_cname	(Shishi *handle, Shishi_asn1 kdcrep, Shishi_asn1 encticketpart);
int	shishi_as_check_cname	(Shishi *handle, Shishi_asn1 asreq, Shishi_asn1 asrep);
int	shishi_kdc_copy_nonce	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_asn1 enckdcreppart);
int	shishi_kdc_check_nonce	(Shishi *handle, Shishi_asn1 kdcreq, Shishi_asn1 enckdcreppart);
Shishi_asn1	shishi_asrep	(Shishi *handle);
Shishi_asn1	shishi_tgsrep	(Shishi *handle);
int	shishi_kdcrep_save	(Shishi *handle, FILE *fh, Shishi_asn1 kdcrep);
int	shishi_kdcrep_print	(Shishi *handle, FILE *fh, Shishi_asn1 kdcrep);
int	shishi_kdcrep_to_file	(Shishi *handle, Shishi_asn1 kdcrep, int filetype, const char *filename);
int	shishi_kdcrep_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *kdcrep);
int	shishi_kdcrep_read	(Shishi *handle, FILE *fh, Shishi_asn1 *kdcrep);
int	shishi_kdcrep_from_file	(Shishi *handle, Shishi_asn1 *kdcrep,

		int filetype, const char *filename);
int	shishi_kdcrep_clear_padata	(Shishi *handle, Shishi_asn1 kdcrep);
int	shishi_kdcrep_get_enc_part_etype	(Shishi *handle, Shishi_asn1 kdcrep, int32_t *etype);
int	shishi_kdcrep_add_enc_part	(Shishi *handle, Shishi_asn1 kdcrep, Shishi_key *key, int keyusage, Shishi_asn1 enckdcreppart);
int	shishi_kdcrep_get_ticket	(Shishi *handle, Shishi_asn1 kdcrep, Shishi_asn1 *ticket);
int	shishi_kdcrep_set_ticket	(Shishi *handle, Shishi_asn1 kdcrep, Shishi_asn1 ticket);
int	shishi_kdcrep_crealm_set	(Shishi *handle, Shishi_asn1 kdcrep, const char *crealm);
int	shishi_kdcrep_cname_set	(Shishi *handle, Shishi_asn1 kdcrep, Shishi_name_type name_type, const char *cname[]);
int	shishi_kdcrep_client_set	(Shishi *handle, Shishi_asn1 kdcrep, const char *client);
int	shishi_kdcrep_crealmserver_set	(Shishi *handle, Shishi_asn1 kdcrep, const char *crealm, const char *client);
int	shishi_kdcrep_set_enc_part	(Shishi *handle, Shishi_asn1 kdcrep, int32_t etype, uint32_t kvno, const char *buf, size_t buflen);
int	shishi_kdcrep_decrypt	(Shishi *handle, Shishi_asn1 kdcrep, Shishi_key *key, int keyusage, Shishi_asn1 *enckdcreppart);
Shishi_asn1	shishi_enckdcreppart	(Shishi *handle);
Shishi_asn1	shishi_encasreppart	(Shishi *handle);
int	shishi_enckdcreppart_get_key	(Shishi *handle, Shishi_asn1 enckdcreppart, Shishi_key **key);
int	shishi_enckdcreppart_key_set	(Shishi *handle, Shishi_asn1 enckdcreppart, Shishi_key *key);
int	shishi_enckdcreppart_nonce_set	(Shishi *handle, Shishi_asn1 enckdcreppart, uint32_t nonce);
int	shishi_enckdcreppart_flags_set	(Shishi *handle, Shishi_asn1 enckdcreppart, int flags);

int	shishi_enckdcreppart_authtime_set	(Shishi *handle, Shishi_asn1 enckdcreppart, const char *authtime);
int	shishi_enckdcreppart_starttime_set	(Shishi *handle, Shishi_asn1 enckdcreppart, const char *starttime);
int	shishi_enckdcreppart_endtime_set	(Shishi *handle, Shishi_asn1 enckdcreppart, const char *endtime);
int	shishi_enckdcreppart_renew_till_set	(Shishi *handle, Shishi_asn1 enckdcreppart, const char *renew_till);
int	shishi_enckdcreppart_srealm_set	(Shishi *handle, Shishi_asn1 enckdcreppart, const char *srealm);
int	shishi_enckdcreppart_sname_set	(Shishi *handle, Shishi_asn1 enckdcreppart, Shishi_name_type name_type, char *sname[]);
int	shishi_enckdcreppart_server_set	(Shishi *handle, Shishi_asn1 enckdcreppart, const char *server);
int	shishi_enckdcreppart_srealmserver_set	(Shishi *handle, Shishi_asn1 enckdcreppart, const char *srealm, const char *server);
int	shishi_enckdcreppart_populate_entticketpart	(Shishi *handle, Shishi_asn1 enckdcreppart, Shishi_asn1 entticketpart);
Shishi_asn1	shishi_krberror	(Shishi *handle);
int	shishi_krberror_print	(Shishi *handle, FILE *fh, Shishi_asn1 krberror);
int	shishi_krberror_save	(Shishi *handle, FILE *fh, Shishi_asn1 krberror);
int	shishi_krberror_to_file	(Shishi *handle, Shishi_asn1 krberror, int filetype, const char *filename);
int	shishi_krberror_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *krberror);
int	shishi_krberror_read	(Shishi *handle, FILE *fh, Shishi_asn1 *krberror);
int	shishi_krberror_from_file	(Shishi *handle, Shishi_asn1 *krberror, int filetype, const char *filename);
int	shishi_krberror_build	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_der	(Shishi *handle, Shishi_asn1 krberror, char **out,

		size_t *outlen);
int	shishi_krberror_crealm	(Shishi *handle, Shishi_asn1 krberror, char **realm, size_t *realmrlen);
int	shishi_krberror_remove_crealm	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_set_crealm	(Shishi *handle, Shishi_asn1 krberror, const char *crealm);
int	shishi_krberror_client	(Shishi *handle, Shishi_asn1 krberror, char **client, size_t *clientrlen);
int	shishi_krberror_set_cname	(Shishi *handle, Shishi_asn1 krberror, Shishi_name_type name_type, const char *cname[]);
int	shishi_krberror_remove_cname	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_client_set	(Shishi *handle, Shishi_asn1 krberror, const char *client);
int	shishi_krberror_realm	(Shishi *handle, Shishi_asn1 krberror, char **realm, size_t *realmrlen);
int	shishi_krberror_set_realm	(Shishi *handle, Shishi_asn1 krberror, const char *realm);
int	shishi_krberror_server	(Shishi *handle, Shishi_asn1 krberror, char **server, size_t *serverrlen);
int	shishi_krberror_remove_sname	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_set_sname	(Shishi *handle, Shishi_asn1 krberror, Shishi_name_type name_type, const char *sname[]);
int	shishi_krberror_server_set	(Shishi *handle, Shishi_asn1 krberror, const char *server);
int	shishi_krberror_ctime	(Shishi *handle, Shishi_asn1 krberror, char **t);
int	shishi_krberror_ctime_set	(Shishi *handle, Shishi_asn1 krberror, const char *t);
int	shishi_krberror_remove_ctime	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_cusec	(Shishi *handle, Shishi_asn1 krberror, uint32_t *cusec);
int	shishi_krberror_cusec_set	(Shishi *handle, Shishi_asn1 krberror, uint32_t cusec);

int	shishi_krberror_remove_cusec	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_stime	(Shishi *handle, Shishi_asn1 krberror, char **t);
int	shishi_krberror_stime_set	(Shishi *handle, Shishi_asn1 krberror, const char *t);
int	shishi_krberror_susec	(Shishi *handle, Shishi_asn1 krberror, uint32_t *susec);
int	shishi_krberror_susec_set	(Shishi *handle, Shishi_asn1 krberror, uint32_t susec);
int	shishi_krberror_errorcode_set	(Shishi *handle, Shishi_asn1 krberror, int errorcode);
int	shishi_krberror_etext	(Shishi *handle, Shishi_asn1 krberror, char **etext, size_t *etextlen);
int	shishi_krberror_set_etext	(Shishi *handle, Shishi_asn1 krberror, const char *etext);
int	shishi_krberror_remove_etext	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_edata	(Shishi *handle, Shishi_asn1 krberror, char **edata, size_t *edatalen);
int	shishi_krberror_set_edata	(Shishi *handle, Shishi_asn1 krberror, const char *edata);
int	shishi_krberror_remove_edata	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_errorcode	(Shishi *handle, Shishi_asn1 krberror, int *errorcode);
int	shishi_krberror_errorcode_fast	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_pretty_print	(Shishi *handle, FILE *fh, Shishi_asn1 krberror);
const char *	shishi_krberror_errorcode_message	(Shishi *handle, int errorcode);
const char *	shishi_krberror_message	(Shishi *handle, Shishi_asn1 krberror);
int	shishi_krberror_methoddata	(Shishi *handle, Shishi_asn1 krberror, Shishi_asn1 *methoddata);
const char *	shishi_generalize_time	(Shishi *handle, time_t t);
const char *	shishi_generalize_now	(Shishi *handle);
time_t	shishi_generalize_ctime	(Shishi *handle, const char *t);
int	shishi_time	(Shishi *handle, Shishi_asn1 node,

		const char *field, char **t);
int	shishi_ctime	(Shishi *handle, Shishi_asn1 node, const char *field, time_t *t);
int	shishi_randomize	(Shishi *handle, int strong, void *data, size_t datalen);
int	shishi_crc	(Shishi *handle, const char *in, size_t inlen, char *out[4]);
int	shishi_md4	(Shishi *handle, const char *in, size_t inlen, char *out[16]);
int	shishi_md5	(Shishi *handle, const char *in, size_t inlen, char *out[16]);
int	shishi_hmac_md5	(Shishi *handle, const char *key, size_t keylen, const char *in, size_t inlen, char *outhash[16]);
int	shishi_hmac_shal	(Shishi *handle, const char *key, size_t keylen, const char *in, size_t inlen, char *outhash[20]);
int	shishi_des_cbc_mac	(Shishi *handle, const char key[8], const char iv[8], const char *in, size_t inlen, char *out[8]);
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);
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);

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);
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);
int	shishi_cipher_supported_p	(int type);
const char *	shishi_cipher_name	(int type);
int	shishi_cipher_blocksize	(int type);
int	shishi_cipher_confoundersize	(int type);
size_t	shishi_cipher_keylen	(int type);
size_t	shishi_cipher_randomlen	(int type);
int	shishi_cipher_defaulttcksumtype	(int32_t type);
int	shishi_cipher_parse	(const char *cipher);
int	shishi_checksum_supported_p	(int32_t type);
const char *	shishi_checksum_name	(int32_t type);
size_t	shishi_checksum_cksumlen	(int32_t type);
int	shishi_checksum_parse	(const char *checksum);
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);
int	shishi_random_to_key	(Shishi *handle, int32_t keytype, const char *rnd, size_t rndlen, Shishi_key *outkey);
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);
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);
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);
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);
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);
int	shishi_encrypt	(Shishi *handle, Shishi_key *key, int keyusage, char *in, size_t inlen, char **out, size_t *outlen);
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);
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);
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);
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);
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);
int	shishi_decrypt	(Shishi *handle, Shishi_key *key, int keyusage, const char *in, size_t inlen, char **out, size_t *outlen);
int	shishi_checksum	(Shishi *handle, Shishi_key *key, int keyusage, int32_t cksumtype, const char *in, size_t inlen, char **out, size_t *outlen);
int	shishi_verify	(Shishi *handle, Shishi_key *key, int keyusage, int cksumtype, const char *in, size_t inlen, const char *cksum,

int	shishi_dk	size_t cksumlen); (Shishi *handle, Shishi_key *key, const char *prfconstant, size_t prfconstantlen, Shishi_key *derivedkey);
int	shishi_dr	(Shishi *handle, Shishi_key *key, const char *prfconstant, size_t prfconstantlen, char *derivedrandom, size_t derivedrandomlen);
int	shishi_n_fold	(Shishi *handle, const char *in, size_t inlen, char *out, size_t outlen);
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);
Shishi_crypto *	shishi_crypto	(Shishi *handle, Shishi_key *key, int keyusage, int32_t etype, const char *iv, size_t ivlen);
void	shishi_crypto_close	(Shishi_crypto *ctx);
int	shishi_crypto_encrypt	(Shishi_crypto *ctx, const char *in, size_t inlen, char **out, size_t *outlen);
int	shishi_crypto_decrypt	(Shishi_crypto *ctx, const char *in, size_t inlen, char **out, size_t *outlen);
const char *	shishi_check_version	(const char *req_version);
int	(*shishi_prompt_password_func)	(Shishi *handle, char **s, const char *format, va_list ap);
void	shishi_prompt_password_callback_set	(Shishi *handle, shishi_prompt_password_func cb);
shishi_prompt_password_func	shishi_prompt_password_callback_get	(Shishi *handle);
int	shishi_prompt_password	(Shishi *handle, char **s, const char *format, ...);
int	shishi_asn1_number_of_elements	(Shishi *handle, Shishi_asn1 node,

		const char *field, size_t *n);
int	shishi_asn1_empty_p	(Shishi *handle, Shishi_asn1 node, const char *field);
int	shishi_asn1_read	(Shishi *handle, Shishi_asn1 node, const char *field, char **data, size_t *datalen);
int	shishi_asn1_read_inline	(Shishi *handle, Shishi_asn1 node, const char *field, char *data, size_t *datalen);
int	shishi_asn1_read_integer	(Shishi *handle, Shishi_asn1 node, const char *field, int *i);
int	shishi_asn1_read_int32	(Shishi *handle, Shishi_asn1 node, const char *field, int32_t *i);
int	shishi_asn1_read_uint32	(Shishi *handle, Shishi_asn1 node, const char *field, uint32_t *i);
int	shishi_asn1_read_bitstring	(Shishi *handle, Shishi_asn1 node, const char *field, uint32_t *flags);
int	shishi_asn1_read_optional	(Shishi *handle, Shishi_asn1 node, const char *field, char **data, size_t *datalen);
int	shishi_asn1_write	(Shishi *handle, Shishi_asn1 node, const char *field, const char *data, size_t datalen);
int	shishi_asn1_write_integer	(Shishi *handle, Shishi_asn1 node, const char *field, int n);
int	shishi_asn1_write_int32	(Shishi *handle, Shishi_asn1 node, const char *field, int32_t n);
int	shishi_asn1_write_uint32	(Shishi *handle, Shishi_asn1 node, const char *field, uint32_t n);
int	shishi_asn1_write_bitstring	(Shishi *handle, Shishi_asn1 node, const char *field, uint32_t flags);

void	shishi_asn1_done	(Shishi *handle, Shishi_asn1 node);
Shishi_asn1	shishi_asn1_pa_enc_ts_enc	(Shishi *handle);
Shishi_asn1	shishi_asn1_encrypteddata	(Shishi *handle);
Shishi_asn1	shishi_asn1_padata	(Shishi *handle);
Shishi_asn1	shishi_asn1_methoddata	(Shishi *handle);
Shishi_asn1	shishi_asn1_etype_info	(Shishi *handle);
Shishi_asn1	shishi_asn1_etype_info2	(Shishi *handle);
Shishi_asn1	shishi_asn1_asreq	(Shishi *handle);
Shishi_asn1	shishi_asn1_asrep	(Shishi *handle);
Shishi_asn1	shishi_asn1_tgsreq	(Shishi *handle);
Shishi_asn1	shishi_asn1_tgsrep	(Shishi *handle);
Shishi_asn1	shishi_asn1_apreq	(Shishi *handle);
Shishi_asn1	shishi_asn1_aprep	(Shishi *handle);
Shishi_asn1	shishi_asn1_ticket	(Shishi *handle);
Shishi_asn1	shishi_asn1_encapreppart	(Shishi *handle);
Shishi_asn1	shishi_asn1_encticketpart	(Shishi *handle);
Shishi_asn1	shishi_asn1_authenticator	(Shishi *handle);
Shishi_asn1	shishi_asn1_enckdcreppart	(Shishi *handle);
Shishi_asn1	shishi_asn1_encasreppart	(Shishi *handle);
Shishi_asn1	shishi_asn1_krberror	(Shishi *handle);
Shishi_asn1	shishi_asn1_krbsafe	(Shishi *handle);
Shishi_asn1	shishi_asn1_priv	(Shishi *handle);
Shishi_asn1	shishi_asn1_encprivpart	(Shishi *handle);
int	shishi_asn1_to_der	(Shishi *handle, Shishi_asn1 node, char **der, size_t *len);
int	shishi_asn1_to_der_field	(Shishi *handle, Shishi_asn1 node, const char *field, char **der, size_t *len);
Shishi_msgtype	shishi_asn1_msgtype	(Shishi *handle, Shishi_asn1 node);
Shishi_msgtype	shishi_der_msgtype	(Shishi *handle, const char *der, size_t derlen);
void	shishi_asn1_print	(Shishi *handle, Shishi_asn1 node, FILE *fh);
Shishi_asn1	shishi_der2asn1	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_padata	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_methoddata	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_etype_info	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_etype_info2	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_ticket	(Shishi *handle,

Shishi_asn1	shishi_der2asn1_encticketpart	const char *der, size_t derlen); (Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_asreq	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_tgsreq	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_asrep	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_tgsrep	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_kdcrep	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_kdcreq	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_apreq	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_aprep	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_authenticator	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_krberror	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_krbsafe	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_priv	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_encasreppart	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_enctgsreppart	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_enckdcreppart	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_encapreppart	(Shishi *handle, const char *der, size_t derlen);
Shishi_asn1	shishi_der2asn1_encprivpart	(Shishi *handle, const char *der, size_t derlen);
int	shishi_ap	(Shishi *handle,

int	shishi_ap_etype	Shishi_ap **ap); (Shishi *handle, Shishi_ap **ap, int etype);
int	shishi_ap_nosubkey	(Shishi *handle, Shishi_ap **ap);
void	shishi_ap_done	(Shishi_ap *ap);
int	shishi_ap_set_tktoptions	(Shishi_ap *ap, Shishi_tkt *tkt, int options);
int	shishi_ap_tktoptions	(Shishi *handle, Shishi_ap **ap, Shishi_tkt *tkt, int options);
int	shishi_ap_etype_tktoptionsdata	(Shishi *handle, Shishi_ap **ap, int32_t etype, Shishi_tkt *tkt, int options, const char *data, size_t len);
int	shishi_ap_set_tktoptionsdata	(Shishi_ap *ap, Shishi_tkt *tkt, int options, const char *data, size_t len);
int	shishi_ap_tktoptionsdata	(Shishi *handle, Shishi_ap **ap, Shishi_tkt *tkt, int options, const char *data, size_t len);
int	shishi_ap_set_tktoptionsraw	(Shishi_ap *ap, Shishi_tkt *tkt, int options, int32_t cksumtype, const char *data, size_t len);
int	shishi_ap_tktoptionsraw	(Shishi *handle, Shishi_ap **ap, Shishi_tkt *tkt, int options, int32_t cksumtype, const char *data, size_t len);
int	shishi_ap_set_tktoptionsasnusage	(Shishi_ap *ap, Shishi_tkt *tkt, int options, Shishi_asn1 node, const char *field, int authenticatorcksumkeyusage, int authenticatorkeyusage);
int	shishi_ap_tktoptionsasnusage	(Shishi *handle, Shishi_ap **ap, Shishi_tkt *tkt, int options, Shishi_asn1 node,

		const char *field, int authenticatorcksumkeyusage, int authenticatorkeyusage);
Shishi_tkt *	shishi_ap_tkt	(Shishi_ap *ap);
void	shishi_ap_tkt_set	(Shishi_ap *ap, Shishi_tkt *tk);
int	shishi_ap_authenticator_cksumdata	(Shishi_ap *ap, char *out, size_t *len);
void	shishi_ap_authenticator_cksumdata_set	(Shishi_ap *ap, const char *authenticatorcksumdat size_t authenticatorcksumdatalen)
void	shishi_ap_authenticator_cksumraw_set	(Shishi_ap *ap, int32_t authenticatorcksumtype, const char *authenticatorcksumraw size_t authenticatorcksumrawlen);
int32_t	shishi_ap_authenticator_cksumtype	(Shishi_ap *ap);
void	shishi_ap_authenticator_cksumtype_set	(Shishi_ap *ap, int32_t cksumtype);
Shishi_asn1	shishi_ap_authenticator	(Shishi_ap *ap);
void	shishi_ap_authenticator_set	(Shishi_ap *ap, Shishi_asn1 authenticator);
Shishi_asn1	shishi_ap_req	(Shishi_ap *ap);
void	shishi_ap_req_set	(Shishi_ap *ap, Shishi_asn1 apreq);
int	shishi_ap_req_der	(Shishi_ap *ap, char **out, size_t *outlen);
int	shishi_ap_req_der_set	(Shishi_ap *ap, char *der, size_t derlen);
int	shishi_ap_req_build	(Shishi_ap *ap);
int	shishi_ap_req_asn1	(Shishi_ap *ap, Shishi_asn1 *apreq);
Shishi_key *	shishi_ap_key	(Shishi_ap *ap);
int	shishi_ap_req_decode	(Shishi_ap *ap);
int	shishi_ap_req_process	(Shishi_ap *ap, Shishi_key *key);
int	shishi_ap_req_process_keyusage	(Shishi_ap *ap, Shishi_key *key, int32_t keyusage);
Shishi_asn1	shishi_ap_rep	(Shishi_ap *ap);
void	shishi_ap_rep_set	(Shishi_ap *ap, Shishi_asn1 aprep);
int	shishi_ap_rep_der	(Shishi_ap *ap, char **out, size_t *outlen);
int	shishi_ap_rep_der_set	(Shishi_ap *ap, char *der, size_t derlen);
int	shishi_ap_rep_verify	(Shishi_ap *ap);
int	shishi_ap_rep_verify_der	(Shishi_ap *ap, char *der, size_t derlen);

int	shishi_ap_rep_verify_asn1	(Shishi_ap *ap, Shishi_asn1 aprep);
int	shishi_ap_rep_asn1	(Shishi_ap *ap, Shishi_asn1 *aprep);
int	shishi_ap_rep_build	(Shishi_ap *ap);
Shishi_asn1	shishi_ap_encapreppart	(Shishi_ap *ap);
void	shishi_ap_encapreppart_set	(Shishi_ap *ap, Shishi_asn1 encapreppart);
const char *	shishi_ap_option2string	(Shishi_apoptions option);
Shishi_apoptions	shishi_ap_string2option	(const char *str);
const char *	shishi_key_principal	(const Shishi_key *key);
void	shishi_key_principal_set	(Shishi_key *key, const char *principal);
const char *	shishi_key_realm	(const Shishi_key *key);
void	shishi_key_realm_set	(Shishi_key *key, const char *realm);
int	shishi_key_type	(const Shishi_key *key);
void	shishi_key_type_set	(Shishi_key *key, int32_t type);
const char *	shishi_key_value	(const Shishi_key *key);
void	shishi_key_value_set	(Shishi_key *key, const char *value);
const char *	shishi_key_name	(Shishi_key *key);
size_t	shishi_key_length	(const Shishi_key *key);
uint32_t	shishi_key_version	(const Shishi_key *key);
void	shishi_key_version_set	(Shishi_key *key, uint32_t kvno);
time_t	shishi_key_timestamp	(const Shishi_key *key);
void	shishi_key_timestamp_set	(Shishi_key *key, time_t timestamp);
int	shishi_key	(Shishi *handle, Shishi_key **key);
void	shishi_key_done	(Shishi_key *key);
void	shishi_key_copy	(Shishi_key *dstkey, Shishi_key *srckey);
int	shishi_key_print	(Shishi *handle, FILE *fh, const Shishi_key *key);
int	shishi_key_to_file	(Shishi *handle, const char *filename, Shishi_key *key);
int	shishi_key_parse	(Shishi *handle, FILE *fh, Shishi_key **key);
int	shishi_key_random	(Shishi *handle, int32_t type, Shishi_key **key);
int	shishi_key_from_value	(Shishi *handle, int32_t type, const char *value, Shishi_key **key);
int	shishi_key_from_base64	(Shishi *handle, int32_t type, const char *value, Shishi_key **key);
int	shishi_key_from_random	(Shishi *handle, int32_t type,

		const char *rnd, size_t rndlen, Shishi_key **outkey);
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);
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);
int	shishi_keys	(Shishi *handle, Shishi_keys **keys);
void	shishi_keys_done	(Shishi_keys **keys);
int	shishi_keys_size	(Shishi_keys *keys);
const Shishi_key *	shishi_keys_nth	(Shishi_keys *keys, int keyno);
void	shishi_keys_remove	(Shishi_keys *keys, int keyno);
int	shishi_keys_add	(Shishi_keys *keys, Shishi_key *key);
int	shishi_keys_print	(Shishi_keys *keys, FILE *fh);
int	shishi_keys_from_file	(Shishi_keys *keys, const char *filename);
int	shishi_keys_to_file	(Shishi *handle, const char *filename, Shishi_keys *keys);
Shishi_key *	shishi_keys_for_serverrealm_in_file	(Shishi *handle, const char *filename, const char *server, const char *realm);
Shishi_key *	shishi_keys_for_server_in_file	(Shishi *handle, const char *filename, const char *server);
Shishi_key *	shishi_keys_for_localservicerealm_in_file	(Shishi *handle, const char *filename, const char *service, const char *realm);
int	shishi_keys_add_keytab_mem	(Shishi *handle, const char *data, size_t len, Shishi_keys *keys);
int	shishi_keys_add_keytab_file	(Shishi *handle, const char *filename, Shishi_keys *keys);
int	shishi_keys_from_keytab_mem	(Shishi *handle, const char *data, size_t len,

int	shishi_keys_from_keytab_file	Shishi_keys **outkeys); (Shishi *handle, const char *filename, Shishi_keys **outkeys);
int	shishi_keys_to_keytab_mem	(Shishi *handle, Shishi_keys *keys, char **out, size_t *len);
int	shishi_keys_to_keytab_file	(Shishi *handle, Shishi_keys *keys, const char *filename);
const char *	shishi_hostkeys_default_file	(Shishi *handle);
void	shishi_hostkeys_default_file_set	(Shishi *handle, const char *hostkeysfile);
Shishi_key *	shishi_hostkeys_for_server	(Shishi *handle, const char *server);
Shishi_key *	shishi_hostkeys_for_serverrealm	(Shishi *handle, const char *server, const char *realm);
Shishi_key *	shishi_hostkeys_for_localservicerealm	(Shishi *handle, const char *service, const char *realm);
Shishi_key *	shishi_hostkeys_for_localservice	(Shishi *handle, const char *service);
Shishi_asn1	shishi_encapreppart	(Shishi *handle);
int	shishi_encapreppart_time_copy	(Shishi *handle, Shishi_asn1 encapreppart, Shishi_asn1 authenticator);
int	shishi_encapreppart_ctime	(Shishi *handle, Shishi_asn1 encapreppart, char **t);
int	shishi_encapreppart_ctime_set	(Shishi *handle, Shishi_asn1 encapreppart, const char *t);
int	shishi_encapreppart_cusec_get	(Shishi *handle, Shishi_asn1 encapreppart, uint32_t *cusec);
int	shishi_encapreppart_cusec_set	(Shishi *handle, Shishi_asn1 encapreppart, uint32_t cusec);
int	shishi_encapreppart_print	(Shishi *handle, FILE *fh, Shishi_asn1 encapreppart);
int	shishi_encapreppart_save	(Shishi *handle, FILE *fh, Shishi_asn1 encapreppart);
int	shishi_encapreppart_to_file	(Shishi *handle, Shishi_asn1 encapreppart, int filetype, const char *filename);
int	shishi_encapreppart_read	(Shishi *handle, FILE *fh, Shishi_asn1 *encapreppart);
int	shishi_encapreppart_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *encapreppart);

int	shishi_encapreppart_from_file	(Shishi *handle, Shishi_asn1 *encapreppart, int filetype, const char *filename);
int	shishi_encapreppart_get_key	(Shishi *handle, Shishi_asn1 encapreppart, Shishi_key **key);
int	shishi_encapreppart_seqnumber_get	(Shishi *handle, Shishi_asn1 encapreppart, uint32_t *seqnumber);
int	shishi_encapreppart_seqnumber_remove	(Shishi *handle, Shishi_asn1 encapreppart);
int	shishi_encapreppart_seqnumber_set	(Shishi *handle, Shishi_asn1 encapreppart, uint32_t seqnumber);
Shishi_asn1	shishi_apreq	(Shishi *handle);
int	shishi_apreq_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *apreq);
int	shishi_apreq_from_file	(Shishi *handle, Shishi_asn1 *apreq, int filetype, const char *filename);
int	shishi_apreq_print	(Shishi *handle, FILE *fh, Shishi_asn1 apreql);
int	shishi_apreq_to_file	(Shishi *handle, Shishi_asn1 apreql, int filetype, const char *filename);
int	shishi_apreq_read	(Shishi *handle, FILE *fh, Shishi_asn1 *apreq);
int	shishi_apreq_save	(Shishi *handle, FILE *fh, Shishi_asn1 apreql);
int	shishi_apreq_set_ticket	(Shishi *handle, Shishi_asn1 apreql, Shishi_asn1 ticket);
int	shishi_apreq_set_authenticator	(Shishi *handle, Shishi_asn1 apreql, int32_t etype, uint32_t kvno, const char *buf, size_t buflen);
int	shishi_apreq_add_authenticator	(Shishi *handle, Shishi_asn1 apreql, Shishi_key *key, int keyusage, Shishi_asn1 authenticator);
int	shishi_apreq_options	(Shishi *handle, Shishi_asn1 apreql, uint32_t *flags);
int	shishi_apreq_use_session_key_p	(Shishi *handle, Shishi_asn1 apreql);
int	shishi_apreq_mutual_required_p	(Shishi *handle,

		Shishi_asn1 apreq);
int	shishi_apreq_options_set	(Shishi *handle, Shishi_asn1 apreq, uint32_t options);
int	shishi_apreq_options_add	(Shishi *handle, Shishi_asn1 apreq, uint32_t option);
int	shishi_apreq_options_remove	(Shishi *handle, Shishi_asn1 apreq, uint32_t option);
int	shishi_apreq_get_ticket	(Shishi *handle, Shishi_asn1 apreq, Shishi_asn1 *ticket);
int	shishi_apreq_get_authenticator_etype	(Shishi *handle, Shishi_asn1 apreq, int32_t *etype);
int	shishi_apreq_decrypt	(Shishi *handle, Shishi_asn1 apreq, Shishi_key *key, int keyusage, Shishi_asn1 *authenticator);
Shishi_asn1	shishi_aprep	(Shishi *handle);
int	shishi_aprep_print	(Shishi *handle, FILE *fh, Shishi_asn1 aprep);
int	shishi_aprep_save	(Shishi *handle, FILE *fh, Shishi_asn1 aprep);
int	shishi_aprep_to_file	(Shishi *handle, Shishi_asn1 aprep, int filetype, const char *filename);
int	shishi_aprep_read	(Shishi *handle, FILE *fh, Shishi_asn1 *aprep);
int	shishi_aprep_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *aprep);
int	shishi_aprep_from_file	(Shishi *handle, Shishi_asn1 *aprep, int filetype, const char *filename);
int	shishi_aprep_decrypt	(Shishi *handle, Shishi_asn1 aprep, Shishi_key *key, int keyusage, Shishi_asn1 *encapreppart);
int	shishi_aprep_verify	(Shishi *handle, Shishi_asn1 authenticator, Shishi_asn1 encapreppart);
int	shishi_aprep_enc_part_set	(Shishi *handle, Shishi_asn1 aprep, int etype, const char *buf, size_t buflen);
int	shishi_aprep_enc_part_add	(Shishi *handle,

		Shishi_asn1 aprep, Shishi_asn1 encticketpart, Shishi_asn1 encapreppart);
int	shishi_aprep_enc_part_make	(Shishi *handle, Shishi_asn1 aprep, Shishi_asn1 encapreppart, Shishi_asn1 authenticator, Shishi_asn1 encticketpart);
int	shishi_aprep_get_enc_part_etype	(Shishi *handle, Shishi_asn1 aprep, int32_t *etype);
int	shishi_kdc_sendrecv	(Shishi *handle, const char *realm, const char *indata, size_t inlen, char **outdata, size_t *outlen);
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);
Shishi_asn1	shishi_encticketpart	(Shishi *handle);
int	shishi_encticketpart_key_set	(Shishi *handle, Shishi_asn1 encticketpart, Shishi_key *key);
int	shishi_encticketpart_get_key	(Shishi *handle, Shishi_asn1 encticketpart, Shishi_key **key);
int	shishi_encticketpart_crealm	(Shishi *handle, Shishi_asn1 encticketpart, char **crealm, size_t *crealm_len);
int	shishi_encticketpart_crealm_set	(Shishi *handle, Shishi_asn1 encticketpart, const char *realm);
int	shishi_encticketpart_client	(Shishi *handle, Shishi_asn1 encticketpart, char **client, size_t *clientlen);
int	shishi_encticketpart_clientrealm	(Shishi *handle, Shishi_asn1 encticketpart, char **client, size_t *clientlen);
int	shishi_encticketpart_cname_set	(Shishi *handle, Shishi_asn1 encticketpart, Shishi_name_type name_type, const char *principal);
int	shishi_encticketpart_print	(Shishi *handle, FILE *fh, Shishi_asn1 encticketpart);
int	shishi_encticketpart_flags_set	(Shishi *handle, Shishi_asn1 encticketpart, int flags);
int	shishi_encticketpart_transited_set	(Shishi *handle,

		Shishi_asn1 encticketpart, int32_t trtype, const char *trdata, size_t trdatalen);
int	shishi_encticketpart_authtime_set	(Shishi *handle, Shishi_asn1 encticketpart, const char *authtime);
int	shishi_encticketpart_endtime_set	(Shishi *handle, Shishi_asn1 encticketpart, const char *endtime);
int	shishi_encticketpart_authtime	(Shishi *handle, Shishi_asn1 encticketpart, char *authtime, size_t *authtimelen);
time_t	shishi_encticketpart_authctime	(Shishi *handle, Shishi_asn1 encticketpart);
int	shishi_safe	(Shishi *handle, Shishi_safe **safe);
void	shishi_safe_done	(Shishi_safe *safe);
Shishi_key *	shishi_safe_key	(Shishi_safe *safe);
void	shishi_safe_key_set	(Shishi_safe *safe, Shishi_key *key);
Shishi_asn1	shishi_safe_safe	(Shishi_safe *safe);
void	shishi_safe_safe_set	(Shishi_safe *safe, Shishi_asn1 asn1safe);
int	shishi_safe_safe_der	(Shishi_safe *safe, char **out, size_t *outlen);
int	shishi_safe_safe_der_set	(Shishi_safe *safe, char *der, size_t derlen);
int	shishi_safe_print	(Shishi *handle, FILE *fh, Shishi_asn1 safe);
int	shishi_safe_save	(Shishi *handle, FILE *fh, Shishi_asn1 safe);
int	shishi_safe_to_file	(Shishi *handle, Shishi_asn1 safe, int filetype, const char *filename);
int	shishi_safe_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *safe);
int	shishi_safe_read	(Shishi *handle, FILE *fh, Shishi_asn1 *safe);
int	shishi_safe_from_file	(Shishi *handle, Shishi_asn1 *safe, int filetype, const char *filename);
int	shishi_safe_cksum	(Shishi *handle, Shishi_asn1 safe, int32_t *cksumtype, char **cksum, size_t *cksumlen);
int	shishi_safe_set_cksum	(Shishi *handle,

		Shishi_asn1 safe, int32_t cksumtype, const char *cksum, size_t cksumlen);
int	shishi_safe_user_data	(Shishi *handle, Shishi_asn1 safe, char **userdata, size_t *userdatalen);
int	shishi_safe_set_user_data	(Shishi *handle, Shishi_asn1 safe, const char *userdata, size_t userdatalen);
int	shishi_safe_build	(Shishi_safe *safe, Shishi_key *key);
int	shishi_safe_verify	(Shishi_safe *safe, Shishi_key *key);
int	shishi_priv	(Shishi *handle, Shishi_priv **priv);
void	shishi_priv_done	(Shishi_priv *priv);
Shishi_key *	shishi_priv_key	(Shishi_priv *priv);
void	shishi_priv_key_set	(Shishi_priv *priv, Shishi_key *key);
Shishi_asn1	shishi_priv_priv	(Shishi_priv *priv);
void	shishi_priv_priv_set	(Shishi_priv *priv, Shishi_asn1 asn1priv);
int	shishi_priv_priv_der	(Shishi_priv *priv, char **out, size_t *outlen);
int	shishi_priv_priv_der_set	(Shishi_priv *priv, char *der, size_t derlen);
Shishi_asn1	shishi_priv_encprivpart	(Shishi_priv *priv);
void	shishi_priv_encprivpart_set	(Shishi_priv *priv, Shishi_asn1 asnlencprivpart);
int	shishi_priv_encprivpart_der	(Shishi_priv *priv, char **out, size_t *outlen);
int	shishi_priv_encprivpart_der_set	(Shishi_priv *priv, char *der, size_t derlen);
int	shishi_priv_print	(Shishi *handle, FILE *fh, Shishi_asn1 priv);
int	shishi_priv_save	(Shishi *handle, FILE *fh, Shishi_asn1 priv);
int	shishi_priv_to_file	(Shishi *handle, Shishi_asn1 priv, int filetype, const char *filename);
int	shishi_priv_parse	(Shishi *handle, FILE *fh, Shishi_asn1 *priv);
int	shishi_priv_read	(Shishi *handle, FILE *fh, Shishi_asn1 *priv);
int	shishi_priv_from_file	(Shishi *handle,

		Shishi_asn1 *priv, int filetype, const char *filename);
int	shishi_priv_enc_part_etype	(Shishi *handle, Shishi_asn1 priv, int32_t *etype);
int	shishi_priv_set_enc_part	(Shishi *handle, Shishi_asn1 priv, int32_t etype, const char *encpart, size_t encpartlen);
int	shishi_encprivpart_user_data	(Shishi *handle, Shishi_asn1 encprivpart, char **userdata, size_t *userdatalen);
int	shishi_encprivpart_set_user_data	(Shishi *handle, Shishi_asn1 encprivpart, const char *userdata, size_t userdatalen);
int	shishi_priv_build	(Shishi_priv *priv, Shishi_key *key);
int	shishi_priv_process	(Shishi_priv *priv, Shishi_key *key);
int	shishi_authorized_p	(Shishi *handle, Shishi_tkt *tkt, const char *authzname);
int	shishi_authorization_parse	(const char *authorization);
int	shishi_authorize_strcmp	(Shishi *handle, const char *principal, const char *authzname);
int	shishi_authorize_k5login	(Shishi *handle, const char *principal, const char *authzname);
char *	shishi_x509ca_default_file_guess	(Shishi *handle);
void	shishi_x509ca_default_file_set	(Shishi *handle, const char *x509cafile);
const char *	shishi_x509ca_default_file	(Shishi *handle);
char *	shishi_x509cert_default_file_guess	(Shishi *handle);
void	shishi_x509cert_default_file_set	(Shishi *handle, const char *x509certfile);
const char *	shishi_x509cert_default_file	(Shishi *handle);
char *	shishi_x509key_default_file_guess	(Shishi *handle);
void	shishi_x509key_default_file_set	(Shishi *handle, const char *x509keyfile);
const char *	shishi_x509key_default_file	(Shishi *handle);
time_t	shishi_get_date	(const char *p, const time_t *now);
void	shishi_xalloc_die	(void);
Shishi_dns	shishi_resolve	(const char *zone, uint16_t querytype);
void	shishi_resolve_free	(Shishi_dns rrs);

Description

Details

SHISHI_VERSION

```
#define SHISHI_VERSION "1.0.0"
```

enum Shishi_rc

```
typedef enum
{
    SHISHI_OK = 0,
    SHISHI_ASN1_ERROR = 1,
    SHISHI_FOPEN_ERROR = 2,
    SHISHI_IO_ERROR = 3,
    SHISHI_MALLOC_ERROR = 4,
    SHISHI_BASE64_ERROR = 5,
    SHISHI_REALM_MISMATCH = 6,
    SHISHI_CNAME_MISMATCH = 7,
    SHISHI_NONCE_MISMATCH = 8,
    SHISHI_TGSREP_BAD_KEYTYPE = 9,
    SHISHI_KDCREP_BAD_KEYTYPE = 10,
    SHISHI_APREP_BAD_KEYTYPE = 11,
    SHISHI_APREP_VERIFY_FAILED = 12,
    SHISHI_APREQ_BAD_KEYTYPE = 13,
    SHISHI_TOO_SMALL_BUFFER = 14,
    SHISHI_DERIVEDKEY_TOO_SMALL = 15,
    SHISHI_KEY_TOO_LARGE = 16,
    SHISHI_CRYPT_ERROR = 17,
    SHISHI_CRYPT_INTERNAL_ERROR = 18,
    SHISHI_SOCKET_ERROR = 19,
    SHISHI_BIND_ERROR = 20,
    SHISHI_SENDTO_ERROR = 21,
    SHISHI_RECVFROM_ERROR = 22,
    SHISHI_CLOSE_ERROR = 23,
    SHISHI_KDC_TIMEOUT = 24,
    SHISHI_KDC_NOT_KNOWN_FOR_REALM = 25,
    SHISHI_TTY_ERROR = 26,
    SHISHI_GOT_KRBERROR = 27,
    SHISHI_HANDLE_ERROR = 28,
    SHISHI_INVALID_TKTS = 29,
    SHISHI_TICKET_BAD_KEYTYPE = 30,
    SHISHI_INVALID_KEY = 31,
    SHISHI_APREQ_DECRYPT_FAILED = 32,
    SHISHI_TICKET_DECRYPT_FAILED = 33,
    SHISHI_INVALID_TICKET = 34,
    SHISHI_OUT_OF_RANGE = 35,
    SHISHI_ASN1_NO_ELEMENT = 36,
    SHISHI_SAFE_BAD_KEYTYPE = 37,
    SHISHI_SAFE_VERIFY_FAILED = 38,
    SHISHI_PKCS5_INVALID_PRF = 39,
    SHISHI_PKCS5_INVALID_ITERATION_COUNT = 40,
    SHISHI_PKCS5_INVALID_DERIVED_KEY_LENGTH = 41,
    SHISHI_PKCS5_DERIVED_KEY_TOO_LONG = 42,
    SHISHI_INVALID_PRINCIPAL_NAME = 43,
    SHISHI_INVALID_ARGUMENT = 44,
    SHISHI_ASN1_NO_VALUE = 45,
    SHISHI_CONNECT_ERROR = 46,
```

```
SHISHI_VERIFY_FAILED = 47,  
SHISHI_PRIV_BAD_KEYTYPE = 48,  
SHISHI_FILE_ERROR = 49,  
SHISHI_ENCAPREPPART_BAD_KEYTYPE = 50,  
SHISHI_GETTIMEOFDAY_ERROR = 51,  
SHISHI_KEYTAB_ERROR = 52,  
SHISHI_CCACHE_ERROR = 53,  
SHISHI_LAST_ERROR = 53  
}  
Shishi_rc;
```

enum Shishi_name_type

```
typedef enum  
{  
    /* Name type not known */  
    SHISHI_NT_UNKNOWN = 0,  
    /* Just the name of the principal as in DCE, or for users */  
    SHISHI_NT_PRINCIPAL = 1,  
    /* Service and other unique instance (krbtgt) */  
    SHISHI_NT_SRV_INST = 2,  
    /* Service with host name as instance (telnet, rcommands) */  
    SHISHI_NT_SRV_HST = 3,  
    /* Service with host as remaining components */  
    SHISHI_NT_SRV_XHST = 4,  
    /* Unique ID */  
    SHISHI_NT_UID = 5,  
    /* Encoded X.509 Distinguished name [RFC 2253] */  
    SHISHI_NT_X500_PRINCIPAL = 6,  
    /* Name in form of SMTP email name (e.g. user@foo.com) */  
    SHISHI_NT_SMTP_NAME = 7,  
    /* Enterprise name - may be mapped to principal name */  
    SHISHI_NT_ENTERPRISE = 10  
}  
Shishi_name_type;
```

enum Shishi_padata_type

```
typedef enum  
{  
    SHISHI_PA_TGS_REQ = 1,  
    SHISHI_PA_ENC_TIMESTAMP = 2,  
    SHISHI_PA_PW_SALT = 3,  
    SHISHI_PA_RESERVED = 4,  
    SHISHI_PA_ENC_UNIX_TIME = 5, /* (deprecated) */  
    SHISHI_PA_SANDIA_SECUREID = 6,  
    SHISHI_PA_SESAME = 7,  
    SHISHI_PA_OSF_DCE = 8,  
    SHISHI_PA_CYBERSAFE_SECUREID = 9,  
    SHISHI_PA_AFS3_SALT = 10,  
    SHISHI_PA_ETYPE_INFO = 11,  
    SHISHI_PA_SAM_CHALLENGE = 12, /* (sam/otp) */  
    SHISHI_PA_SAM_RESPONSE = 13, /* (sam/otp) */  
    SHISHI_PA_PK_AS_REQ = 14, /* (pkinit) */  
    SHISHI_PA_PK_AS_REP = 15, /* (pkinit) */  
    SHISHI_PA_ETYPE_INFO2 = 19, /* (replaces pa_etype_info) */  
    SHISHI_PA_USE_SPECIFIED_KVNO = 20,  
    SHISHI_PA_SAM_REDIRECT = 21, /* (sam/otp) */  
}
```

```
SHISHI_PA_GET_FROM_TYPED_DATA = 22,~/* (embedded in typed data) */
SHISHI_TD_PADATA = 22,~/* (embeds padata) */
SHISHI_PA_SAM_ETYPE_INFO = 23,~/* (sam/otp) */
SHISHI_PA_ALT_PRINC = 24,~/* (crawdada@fnal.gov) */
SHISHI_PA_SAM_CHALLENGE2 = 30,~/* (kenh@pobox.com) */
SHISHI_PA_SAM_RESPONSE2 = 31,~/* (kenh@pobox.com) */
SHISHI_PA_EXTRA_TGT = 41,~/* Reserved extra TGT */
SHISHI_TD_PKINIT_CMS_CERTIFICATES = 101,~/* CertificateSet from CMS */
SHISHI_TD_KRB_PRINCIPAL = 102,~/* PrincipalName */
SHISHI_TD_KRB_REALM = 103,~/* Realm */
SHISHI_TD_TRUSTED_CERTIFIERS = 104,~/* from PKINIT */
SHISHI_TD_CERTIFICATE_INDEX = 105,~/* from PKINIT */
SHISHI_TD_APP_DEFINED_ERROR = 106,~/* application specific */
SHISHI_TD_REQ_NONCE = 107,~/* INTEGER */
SHISHI_TD_REQ_SEQ = 108,~/* INTEGER */
SHISHI_PA_PAC_REQUEST = 128~/* (jbrezak@exchange.microsoft.com) */
}
Shishi_padata_type;
```

enum Shishi_tr_type

```
typedef enum
{
    SHISHI_TR_DOMAIN_X500_COMPRESS = 1
}
Shishi_tr_type;
```

enum Shishi_apoptions

```
typedef enum
{
    SHISHI_APOPTIONS_RESERVED = 0x1,~/* bit 0 */
    SHISHI_APOPTIONS_USE_SESSION_KEY = 0x2,~/* bit 1 */
    SHISHI_APOPTIONS_MUTUAL_REQUIRED = 0x4~/* bit 2 */
}
Shishi_apoptions;
```

enum Shishi_ticketflags

```
typedef enum
{
    SHISHI_TICKETFLAGS_RESERVED = 0x1,~/* bit 0 */
    SHISHI_TICKETFLAGS_FORWARDABLE = 0x2,~/* bit 1 */
    SHISHI_TICKETFLAGS_FORWARDED = 0x4,~/* bit 2 */
    SHISHI_TICKETFLAGS_PROXIABLE = 0x8,~/* bit 3 */
    SHISHI_TICKETFLAGS_PROXY = 0x10,~/* bit 4 */
    SHISHI_TICKETFLAGS_MAY_POSTDATE = 0x20,~/* bit 5 */
    SHISHI_TICKETFLAGS_POSTDATED = 0x40,~/* bit 6 */
    SHISHI_TICKETFLAGS_INVALID = 0x80,~/* bit 7 */
    SHISHI_TICKETFLAGS_RENEWABLE = 0x100,~/* bit 8 */
    SHISHI_TICKETFLAGS_INITIAL = 0x200,~/* bit 9 */
    SHISHI_TICKETFLAGS_PRE_AUTHENT = 0x400,~/* bit 10 */
    SHISHI_TICKETFLAGS_HW_AUTHENT = 0x800,~/* bit 11 */
    SHISHI_TICKETFLAGS_TRANSITED_POLICY_CHECKED = 0x1000,~/* bit 12 */
    SHISHI_TICKETFLAGS_OK_AS_DELEGATE = 0x2000~/* bit 13 */
}
Shishi_ticketflags;
```

enum Shishi_KDCOptions

```
typedef enum
{
    SHISHI_KDCOPTIONS_RESERVED = 0x1, /* bit 0 */
    SHISHI_KDCOPTIONS_FORWARDABLE = 0x2, /* bit 1 */
    SHISHI_KDCOPTIONS_FORWARDED = 0x4, /* bit 2 */
    SHISHI_KDCOPTIONS_PROXIABLE = 0x8, /* bit 3 */
    SHISHI_KDCOPTIONS_PROXY = 0x10, /* bit 4 */
    SHISHI_KDCOPTIONS_ALLOW_POSTDATE = 0x20, /* bit 5 */
    SHISHI_KDCOPTIONS_POSTDATED = 0x40, /* bit 6 */
    SHISHI_KDCOPTIONS_UNUSED7 = 0x80, /* bit 7 */
    SHISHI_KDCOPTIONS_RENEWABLE = 0x100, /* bit 8 */
    SHISHI_KDCOPTIONS_UNUSED9 = 0x200, /* bit 9 */
    SHISHI_KDCOPTIONS_UNUSED10 = 0x400, /* bit 10 */
    SHISHI_KDCOPTIONS_UNUSED11 = 0x800, /* bit 11 */
#define SHISHI_KDCOPTIONS_DISABLE_TRANSITED_CHECK 0x4000000 /* bit 26 */
#define SHISHI_KDCOPTIONS_RENEWABLE_OK 0x8000000 /* bit 27 */
#define SHISHI_KDCOPTIONS_ENC_TKT_IN_SKEY 0x10000000 /* bit 28 */
#define SHISHI_KDCOPTIONS_RENEW 0x40000000 /* bit 30 */
#define SHISHI_KDCOPTIONS_VALIDATE 0x80000000 /* bit 31 */
}
Shishi_KDCOptions;
```

enum Shishi_msgtype

```
typedef enum
{
    /* 0 unused */
    /* 1 Ticket PDU */
    /* 2 Authenticator non-PDU */
    /* 3 EncTicketPart non-PDU */
    /* 4-9 unused */
    /* Request for initial authentication */
    SHISHI_MSGTYPE_AS_REQ = 10,
    /* Response to SHISHI_MSGTYPE_AS_REQ request */
    SHISHI_MSGTYPE_AS_REP = 11,
    /* Request for authentication based on TGT */
    SHISHI_MSGTYPE_TGS_REQ = 12,
    /* Response to SHISHI_MSGTYPE_TGS_REQ request */
    SHISHI_MSGTYPE_TGS_REP = 13,
    /* application request to server */
    SHISHI_MSGTYPE_AP_REQ = 14,
    /* Response to SHISHI_MSGTYPE_AP_REQ_MUTUAL */
    SHISHI_MSGTYPE_AP_REP = 15,
    /* Reserved for user-to-user krb_tgt_request */
    SHISHI_MSGTYPE_RESERVED16 = 16,
    /* Reserved for user-to-user krb_tgt_reply */
    SHISHI_MSGTYPE_RESERVED17 = 17,
    /* 18-19 unused */
    /* Safe (checksummed) application message */
    SHISHI_MSGTYPE_SAFE = 20,
    /* Private (encrypted) application message */
    SHISHI_MSGTYPE_PRIV = 21,
    /* Private (encrypted) message to forward credentials */
    SHISHI_MSGTYPE_CRED = 22,
    /* 23-24 unused */
    /* 25 EncASRepPart non-PDU */
    /* 26 EncTGSRepPart non-PDU */
    /* 27 EncApRepPart non-PDU */
}
```

```
/* 28          EncKrbPrivPart non-PDU */
/* 29          EncKrbCredPart non-PDU */
/* Error response */
SHISHI_MSGTYPE_ERROR = 30
}
Shishi_msgtype;
```

enum Shishi_lrtype

```
typedef enum
{
    SHISHI_LRTYPE_LAST_INITIAL_TGT_REQUEST = 1,
    SHISHI_LRTYPE_LAST_INITIAL_REQUEST = 2,
    SHISHI_LRTYPE_NEWEST_TGT_ISSUE = 3,
    SHISHI_LRTYPE_LAST_RENEWAL = 4,
    SHISHI_LRTYPE_LAST_REQUEST = 5
}
Shishi_lrtype;
```

enum Shishi_etype

```
typedef enum
{
    SHISHI_NULL = 0,
    SHISHI_DES_CBC_CRC = 1,
    SHISHI_DES_CBC_MD4 = 2,
    SHISHI_DES_CBC_MD5 = 3,
    SHISHI_DES_CBC_NONE = 4,
    SHISHI_DES3_CBC_NONE = 6,
    SHISHI_DES3_CBC_HMAC_SHA1_KD = 16,
    SHISHI_AES128_CTS_HMAC_SHA1_96 = 17,
    SHISHI_AES256_CTS_HMAC_SHA1_96 = 18,
    SHISHI_ARCFOUR_HMAC = 23,
    SHISHI_ARCFOUR_HMAC_EXP = 24
}
Shishi_etype;
```

enum Shishi_cksumtype

```
typedef enum
{
    SHISHI_CRC32 = 1,
    SHISHI_RSA_MD4 = 2,
    SHISHI_RSA_MD4_DES = 3,
    SHISHI_DES_MAC = 4,
    SHISHI_DES_MAC_K = 5,
    SHISHI_RSA_MD4_DES_K = 6,
    SHISHI_RSA_MD5 = 7,
    SHISHI_RSA_MD5_DES = 8,
    SHISHI_RSA_MD5_DES_GSS = 9, /* XXX */
    SHISHI_HMAC_SHA1_DES3_KD = 12,
    SHISHI_HMAC_SHA1_96_AES128 = 15,
    SHISHI_HMAC_SHA1_96_AES256 = 16,
    SHISHI_ARCFOUR_HMAC_MD5 = -138,
    SHISHI_KRB5_GSSAPI_CKSUM = 8003,
    SHISHI_NO_CKSUMTYPE = -1
}
```



```
}  
Shishi_cksumtype;
```

enum Shishi_filetype

```
typedef enum  
{  
    SHISHI_FILETYPE_TEXT = 0,  
    SHISHI_FILETYPE_DER,  
    SHISHI_FILETYPE_HEX,  
    SHISHI_FILETYPE_BASE64,  
    SHISHI_FILETYPE_BINARY  
}  
Shishi_filetype;
```

enum Shishi_outputtype

```
typedef enum  
{  
    SHISHI_OUTPUTTYPE_NULL = 0,  
    SHISHI_OUTPUTTYPE_STDERR,  
    SHISHI_OUTPUTTYPE_SYSLOG  
}  
Shishi_outputtype;
```

enum Shishi_authorization

```
typedef enum  
{  
    SHISHI_AUTHORIZATION_BASIC = 0,  
    SHISHI_AUTHORIZATION_K5LOGIN  
}  
Shishi_authorization;
```

enum Shishi_keyusage

```
typedef enum  
{  
    /* 1. AS-REQ PA-ENC-TIMESTAMP padata timestamp, encrypted with the  
       client key */  
    SHISHI_KEYUSAGE_ASREQ_PA_ENC_TIMESTAMP = 1,  
    /* 2. AS-REP Ticket and TGS-REP Ticket (includes TGS session key or  
       application session key), encrypted with the service key */  
    SHISHI_KEYUSAGE_ENCTICKETPART = 2,  
    /* 3. AS-REP encrypted part (includes TGS session key or application  
       session key), encrypted with the client key */  
    SHISHI_KEYUSAGE_ENCASREPPART = 3,  
    /* 4. TGS-REQ KDC-REQ-BODY AuthorizationData, encrypted with the TGS  
       session key */  
    SHISHI_KEYUSAGE_TGSREQ_AUTHORIZATIONDATA_TGS_SESSION_KEY = 4,  
    /* 5. TGS-REQ KDC-REQ-BODY AuthorizationData, encrypted with the TGS  
       authenticator subkey (section 5.4.1) */  
    SHISHI_KEYUSAGE_TGSREQ_AUTHORIZATIONDATA_TGS_AUTHENTICATOR_KEY = 5,  
    /* 6. TGS-REQ PA-TGS-REQ padata AP-REQ Authenticator cksum, keyed with the
```

```

    TGS session key */
SHISHI_KEYUSAGE_TGSREQ_APREQ_AUTHENTICATOR_CKSUM = 6,
/* 7. TGS-REQ PA-TGS-REQ padata AP-REQ Authenticator (includes TGS
   authenticator subkey), encrypted with the TGS session key */
SHISHI_KEYUSAGE_TGSREQ_APREQ_AUTHENTICATOR = 7,
/* 8. TGS-REP encrypted part (includes application session key), encrypted
   with the TGS session key */
SHISHI_KEYUSAGE_ENCTGSREPPART_SESSION_KEY = 8,
/* 9. TGS-REP encrypted part (includes application session key), encrypted
   with the TGS authenticator subkey */
SHISHI_KEYUSAGE_ENCTGSREPPART_AUTHENTICATOR_KEY = 9,
/* 10. AP-REQ Authenticator cksum, keyed with the application
   session key */
SHISHI_KEYUSAGE_APREQ_AUTHENTICATOR_CKSUM = 10,
/* 11. AP-REQ Authenticator (includes application authenticator subkey),
   encrypted with the application session key */
SHISHI_KEYUSAGE_APREQ_AUTHENTICATOR = 11,
/* 12. AP-REP encrypted part (includes application session subkey),
   encrypted with the application session key */
SHISHI_KEYUSAGE_ENCAPREPPART = 12,
/* 13. KRB-PRIV encrypted part, encrypted with a key chosen by the
   application */
SHISHI_KEYUSAGE_KRB_PRIV = 13,
/* 14. KRB-CRED encrypted part, encrypted with a key chosen by the
   application */
SHISHI_KEYUSAGE_KRB_CRED = 14,
/* 15. KRB-SAFE cksum, keyed with a key chosen by the application */
SHISHI_KEYUSAGE_KRB_SAFE = 15,
/* 18. KRB-ERROR checksum (e-cksum) */
SHISHI_KEYUSAGE_KRB_ERROR = 18,
/* 19. AD-KDCIssued checksum (ad-checksum) */
SHISHI_KEYUSAGE_AD_KDCISSUED = 19,
/* 20. Checksum for Mandatory Ticket Extensions */
SHISHI_KEYUSAGE_TICKET_EXTENSION = 20,
/* 21. Checksum in Authorization Data in Ticket Extensions */
SHISHI_KEYUSAGE_TICKET_EXTENSION_AUTHORIZATION = 21,
/* 22-24. Reserved for use in GSSAPI mechanisms derived from RFC 1964.
   (raeburn/MIT) */
SHISHI_KEYUSAGE_GSS_R1 = 22,
SHISHI_KEYUSAGE_GSS_R2 = 23,
SHISHI_KEYUSAGE_GSS_R3 = 24,
/* draft-ietf-krb-wg-gssapi-cfx */
SHISHI_KEYUSAGE_ACCEPTOR_SEAL = 22,
SHISHI_KEYUSAGE_ACCEPTOR_SIGN = 23,
SHISHI_KEYUSAGE_INITIATOR_SEAL = 24,
SHISHI_KEYUSAGE_INITIATOR_SIGN = 25,
/* 16-18,20-21,25-511. Reserved for future use. */
/* 512-1023. Reserved for uses internal implementations. */
/* 1024. Encryption for application use in protocols that
   do not specify key usage values */
/* 1025. Checksums for application use in protocols that
   do not specify key usage values */
/* 1026-2047. Reserved for application use.
   1026,1028,1030,1032,1034 used in KCMD protocol */
SHISHI_KEYUSAGE_KCMD_DES = 1026,
SHISHI_KEYUSAGE_KCMD_INPUT = 1028,
SHISHI_KEYUSAGE_KCMD_OUTPUT = 1030,
SHISHI_KEYUSAGE_KCMD_STDERR_INPUT = 1032,
SHISHI_KEYUSAGE_KCMD_STDERR_OUTPUT = 1034
}
Shishi_keyusage;

```

enum Shishi_krb_error

```
typedef enum
{
    /* No error */
    SHISHI_KDC_ERR_NONE = 0,
    /* Client's entry in database has expired */
    SHISHI_KDC_ERR_NAME_EXP = 1,
    /* Server's entry in database has expired */
    SHISHI_KDC_ERR_SERVICE_EXP = 2,
    /* Requested protocol version number - not supported */
    SHISHI_KDC_ERR_BAD_PVNO = 3,
    /* Client's key encrypted in old master key */
    SHISHI_KDC_ERR_C_OLD_MAST_KVNO = 4,
    /* Server's key encrypted in old master key */
    SHISHI_KDC_ERR_S_OLD_MAST_KVNO = 5,
    /* Client not found in database */
    SHISHI_KDC_ERR_C_PRINCIPAL_UNKNOWN = 6,
    /* Server not found in database */
    SHISHI_KDC_ERR_S_PRINCIPAL_UNKNOWN = 7,
    /* Multiple principal entries in database */
    SHISHI_KDC_ERR_PRINCIPAL_NOT_UNIQUE = 8,
    /* The client or server has a null key */
    SHISHI_KDC_ERR_NULL_KEY = 9,
    /* Ticket not eligible for postdating */
    SHISHI_KDC_ERR_CANNOT_POSTDATE = 10,
    /* Requested start time is later than end time */
    SHISHI_KDC_ERR_NEVER_VALID = 11,
    /* KDC policy rejects request */
    SHISHI_KDC_ERR_POLICY = 12,
    /* KDC cannot accommodate requested option */
    SHISHI_KDC_ERR_BADOPTION = 13,
    /* KDC has no support for encryption type */
    SHISHI_KDC_ERR_ETYPE_NOSUPP = 14,
    /* KDC has no support for checksum type */
    SHISHI_KDC_ERR_SUMTYPE_NOSUPP = 15,
    /* KDC has no support for padata type */
    SHISHI_KDC_ERR_PADATA_TYPE_NOSUPP = 16,
    /* KDC has no support for transited type */
    SHISHI_KDC_ERR_TRTYPE_NOSUPP = 17,
    /* Clients credentials have been revoked */
    SHISHI_KDC_ERR_CLIENT_REVOKED = 18,
    /* Credentials for server have been revoked */
    SHISHI_KDC_ERR_SERVICE_REVOKED = 19,
    /* TGT has been revoked */
    SHISHI_KDC_ERR_TGT_REVOKED = 20,
    /* Client not yet valid - try again later */
    SHISHI_KDC_ERR_CLIENT_NOTYET = 21,
    /* Server not yet valid - try again later */
    SHISHI_KDC_ERR_SERVICE_NOTYET = 22,
    /* Password has expired - change password to reset */
    SHISHI_KDC_ERR_KEY_EXPIRED = 23,
    /* Pre-authentication information was invalid */
    SHISHI_KDC_ERR_PREAUTH_FAILED = 24,
    /* Additional pre-authentication required */
    SHISHI_KDC_ERR_PREAUTH_REQUIRED = 25,
    /* Requested server and ticket don't match */
    SHISHI_KDC_ERR_SERVER_NOMATCH = 26,
    /* Server principal valid for user = 2, user only */
    SHISHI_KDC_ERR_MUST_USE_USER2USER = 27,
    /* KDC Policy rejects transited path */
    SHISHI_KDC_ERR_PATH_NOT_ACCPETED = 28,
```

```
/* A service is not available */
SHISHI_KDC_ERR_SVC_UNAVAILABLE = 29,
/* Integrity check on decrypted field failed */
SHISHI_KRB_AP_ERR_BAD_INTEGRITY = 31,
/* Ticket expired */
SHISHI_KRB_AP_ERR_TKT_EXPIRED = 32,
/* Ticket not yet valid */
SHISHI_KRB_AP_ERR_TKT_NYV = 33,
/* Request is a replay */
SHISHI_KRB_AP_ERR_REPEAT = 34,
/* The ticket isn't for us */
SHISHI_KRB_AP_ERR_NOT_US = 35,
/* Ticket and authenticator don't match */
SHISHI_KRB_AP_ERR_BADMATCH = 36,
/* Clock skew too great */
SHISHI_KRB_AP_ERR_SKEW = 37,
/* Incorrect net address */
SHISHI_KRB_AP_ERR_BADADDR = 38,
/* Protocol version mismatch */
SHISHI_KRB_AP_ERR_BADVERSION = 39,
/* Invalid msg type */
SHISHI_KRB_AP_ERR_MSG_TYPE = 40,
/* Message stream modified */
SHISHI_KRB_AP_ERR_MODIFIED = 41,
/* Message out of order */
SHISHI_KRB_AP_ERR_BADORDER = 42,
/* Specified version of key is not available */
SHISHI_KRB_AP_ERR_BADKEYVER = 44,
/* Service key not available */
SHISHI_KRB_AP_ERR_NOKEY = 45,
/* Mutual authentication failed */
SHISHI_KRB_AP_ERR_MUT_FAIL = 46,
/* Incorrect message direction */
SHISHI_KRB_AP_ERR_BADDIRECTION = 47,
/* Alternative authentication method required */
SHISHI_KRB_AP_ERR_METHOD = 48,
/* Incorrect sequence number in message */
SHISHI_KRB_AP_ERR_BADSEQ = 49,
/* Inappropriate type of checksum in message */
SHISHI_KRB_AP_ERR_INAPP_CKSUM = 50,
/* Policy rejects transited path */
SHISHI_KRB_AP_PATH_NOT_ACCEPTED = 51,
/* Response too big for UDP, retry with TCP */
SHISHI_KRB_ERR_RESPONSE_TOO_BIG = 52,
/* Generic error (description in e-text) */
SHISHI_KRB_ERR_GENERIC = 60,
/* Field is too long for this implementation */
SHISHI_KRB_ERR_FIELD_TOOLONG = 61,
/* Reserved for PKINIT */
SHISHI_KDC_ERROR_CLIENT_NOT_TRUSTED = 62,
/* Reserved for PKINIT */
SHISHI_KDC_ERROR_KDC_NOT_TRUSTED = 63,
/* Reserved for PKINIT */
SHISHI_KDC_ERROR_INVALID_SIG = 64,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_KEY_TOO_WEAK = 65,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_CERTIFICATE_MISMATCH = 66,
/* No TGT available to validate USER-TO-USER */
SHISHI_KRB_AP_ERR_NO_TGT = 67,
/* USER-TO-USER TGT issued different KDC */
SHISHI_KDC_ERR_WRONG_REALM = 68,
```

```
/* Ticket must be for USER-TO-USER */
SHISHI_KRB_AP_ERR_USER_TO_USER_REQUIRED = 69,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_CANT_VERIFY_CERTIFICATE = 70,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_INVALID_CERTIFICATE = 71,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_REVOKED_CERTIFICATE = 72,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_REVOCATION_STATUS_UNKNOWN = 73,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_REVOCATION_STATUS_UNAVAILABLE = 74,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_CLIENT_NAME_MISMATCH = 75,
/* Reserved for PKINIT */
SHISHI_KDC_ERR_KDC_NAME_MISMATCH = 76,
SHISHI_LAST_ERROR_CODE = 76
}
Shishi_krb_error;
```

enum Shishi_tkts_hintflags

```
typedef enum
{
    SHISHI_TKTSHINTFLAGS_ACCEPT_EXPIRED = 1,
    SHISHI_TKTSHINTFLAGS_NON_INTERACTIVE = 2
}
Shishi_tkts_hintflags;
```

Shishi_tkts_hint

```
typedef struct Shishi_tkts_hint Shishi_tkts_hint;
```

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

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)
```

shishi_alloc_fail_function ()

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

shishi ()

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

Initializes the Shishi library, and set up, using [shishi_error_set_outputtype\(\)](#), the library so that future warnings and informational messages are printed to stderr. If this function fails, it may print diagnostic errors to stderr.

Returns : Returns Shishi library handle, or **NULL** on error.

shishi_server ()

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

Initializes the Shishi library, and set up, using [shishi_error_set_outputtype\(\)](#), the library so that future warnings and informational messages are printed to the syslog. If this function fails, it may print diagnostic errors to the syslog.

Returns : Returns 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 calls to shishi functions after this.

If there is a default tkts, it is written to the default tkts file (call [shishi_tkts_default_file_set\(\)](#) to change the default tkts file). If you do not wish to write the default tkts file, close the default tkts with [shishi_tkts_done\(handle, NULL\)](#) before calling this function.

handle : shishi handle as allocated by [shishi_init\(\)](#).

shishi_init ()

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

Create a Shishi library handle, using **shishi()**, and read 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 values, except for `SHISHI_HANDLE_ERROR` which indicates a problem allocating the handle. (The other error conditions comes from reading the files.)

handle : pointer to handle to be created.

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);
```

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

The handle is allocated regardless of return values, except for `SHISHI_HANDLE_ERROR` which indicates a problem allocating the handle. (The other error conditions comes from reading the files.)

handle : pointer to handle to be created.

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);
```

Create a Shishi library handle, using **shishi_server()**, and read the system configuration file. The paths to the system configuration file is decided at compile time, and is `$sysconfdir/shishi.conf`.

The handle is allocated regardless of return values, except for `SHISHI_HANDLE_ERROR` which indicates a problem allocating the handle. (The other error conditions comes from reading the file.)

handle : pointer to handle to be created.

Returns : Returns `SHISHI_OK` iff successful.

shishi_init_server_with_paths ()

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

Create a Shishi library handle, using [shishi_server\(\)](#), and read the system configuration file from specified location. The paths to the system configuration file is decided at compile time, and is \$sysconfdir/shishi.conf. The handle is allocated regardless of return values, except for SHISHI_HANDLE_ERROR which indicates a problem allocating the handle. (The other error conditions comes from reading the file.)

handle : pointer to handle to be created.

systemcfgfile : Filename of system configuration, or NULL.

Returns : Returns SHISHI_OK iff successful.

shishi_cfg ()

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

Configure shishi library with given option.

handle : Shishi library handle create by [shishi_init\(\)](#).

option : string with shishi library option.

Returns : Returns SHISHI_OK if option was valid.

shishi_cfg_from_file ()

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

Configure shishi library using configuration file.

handle : Shishi library handle create by [shishi_init\(\)](#).

cfg : filename to read configuration from.

Returns : Returns [SHISHI_OK](#) iff successful.

shishi_cfg_print ()

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

Print library configuration status, mostly for debugging purposes.

handle : Shishi library handle create by [shishi_init\(\)](#).

fh : file descriptor opened for writing.

Returns : 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 may be overridden by the environment variable SHISHI_CONFIG.

handle : Shishi library handle create by [shishi_init\(\)](#).

Returns : Return system configuration file name.

shishi_cfg_default_userdirectory ()

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

The default user directory (used for, e.g. Shishi ticket cache) is normally computed by appending BASE_DIR ("/shishi") to the content of the environment variable \$HOME, but can be overridden by specifying the complete path in the environment variable SHISHI_HOME.

handle : Shishi library handle create by [shishi_init\(\)](#).

Returns : Return directory with configuration files etc.

shishi_cfg_default_userfile ()

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

Get filename of default user configuration file, typically \$HOME/shishi.conf.

handle : Shishi library handle create by [shishi_init\(\)](#).

Returns : Return user configuration filename.

shishi_cfg_userdirectory_file ()

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

Get the full path to specified *file* in the users' configuration directory.

handle : Shishi library handle create by [shishi_init\(\)](#).

file : basename of file to find in user directory.

Returns : Return full path to given relative filename, relative to the user specific Shishi configuration directory as returned by [shishi_cfg_default_userdirectory\(\)](#) (typically \$HOME/.shishi).

shishi_cfg_clientkdcetype ()

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

Set the etypes variable to the array of preferred client etypes.

handle : Shishi library handle create by [shishi_init\(\)](#).

etypes : output array with encryption types.

Returns : Return the number of encryption types in the array, 0 means none.

shishi_cfg_clientkdcetype_fast ()

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

Extract the default etype from the list of preferred client etypes.

handle : Shishi library handle create by **shishi_init()**.

Returns : Return the default encryption types.

shishi_cfg_clientkdcetype_set ()

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

Set the "client-kdc-etypes" configuration option from given string. The string contains encryption types (integer or names) separated by comma or whitespace, e.g. "aes256-cts-hmac-sha1-96 des3-cbc-sha1-kd des-cbc-md5".

handle : Shishi library handle create by **shishi_init()**.

value : string with encryption types.

Returns : Return SHISHI_OK iff successful.

shishi_cfg_authorizationtype_set ()

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

Set the "authorization-types" configuration option from given string. The string contains authorization types (integer or names) separated by comma or whitespace, e.g. "basic k5login".

handle : Shishi library handle create by **shishi_init()**.

value : string with authorization types.

Returns : Return SHISHI_OK iff successful.

shishi_strerror ()

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

Convert return code to human readable string.

err : shishi error code.

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

shishi_error ()

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

Extract detailed error information string. Note that the memory is managed by the Shishi library, so you must not deallocate the string.

handle : shishi handle as allocated by **shishi_init()**.

Returns : Returns pointer to error information string, that must not be deallocate by caller.

shishi_error_clear ()

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

Clear the detailed error information string. See [shishi_error\(\)](#) for how to access the error string, and [shishi_error_set\(\)](#) and [shishi_error_printf\(\)](#) for how to set the error string. This function is mostly for Shishi internal use, but if you develop an extension of Shishi, it may be useful to use the same error handling infrastructure.

handle : shishi handle as allocated by [shishi_init\(\)](#).

shishi_error_set ()

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

Set the detailed error information string to specified string. The string is copied into the Shishi internal structure, so you can deallocate the string passed to this function after the call. This function is mostly for Shishi internal use, but if you develop an extension of Shishi, it may be useful to use the same error handling infrastructure.

handle : shishi handle as allocated by [shishi_init\(\)](#).

errstr : Zero terminated character array containing error description, or NULL to clear the error description string.

shishi_error_printf ()

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

Set the detailed error information string to a printf formatted string. This function is mostly for Shishi internal use, but if you develop an extension of Shishi, it may be useful to use the same error handling infrastructure.

handle : shishi handle as allocated by [shishi_init\(\)](#).

format : printf style format string.

... : print style arguments.

shishi_error_outputtype ()

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

Get the current output type for logging messages.

handle : shishi handle as allocated by [shishi_init\(\)](#).

Returns : Return output type (NULL, stderr or syslog) for informational and warning messages.

shishi_error_set_outputtype ()

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

Set output type (NULL, stderr or syslog) for informational and warning messages.

handle : shishi handle as allocated by [shishi_init\(\)](#).

type : output type.

shishi_info ()

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

Print informational message to output as defined in handle.

handle : shishi handle as allocated by [shishi_init\(\)](#).

format : printf style format string.

... : print style arguments.

shishi_warn ()

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

Print a warning to output as defined in handle.

handle : shishi handle as allocated by [shishi_init\(\)](#).

format : printf style format string.

... : print style arguments.

shishi_verbose ()

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

Print a diagnostic message to output as defined in handle.

handle : shishi handle as allocated by [shishi_init\(\)](#).

format : printf style format string.

... : print style arguments.

shishi_realm_default_guess ()

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

Guesses a realm based on [getdomainname\(\)](#) (which really is NIS/YP domain, but if it is set it might be a good guess), or if it fails, based on [gethostname\(\)](#), or if it fails, the string "could-not-guess-default-realm". Note that the hostname is not trimmed off of the data returned by [gethostname\(\)](#) to get the domain name and use that as the realm.

Returns : Returns guessed realm for host as a string that has to be deallocated with [free\(\)](#) by the caller.

shishi_realm_default ()

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

Get name of default realm.

handle : Shishi library handle create by **shishi_init()**.

Returns : Returns the default realm used in the library. (Not a copy of it, so don't modify or deallocate it.)

shishi_realm_default_set ()

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

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

handle : Shishi library handle create by **shishi_init()**.

realm : string with new default realm name, or NULL to reset to default.

shishi_realm_for_server_file ()

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

Find realm for a host using configuration file.

handle : Shishi library handle create by **shishi_init()**.

server : hostname to find realm for.

Returns : Returns realm for host, or NULL if not found.

shishi_realm_for_server_dns ()

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

Find realm for a host using DNS lookups, according to draft-ietf-krb-wg-krb-dns-locate-03.txt. Since DNS lookups may 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 cross-realm trust it may redirect you to a compromised realm. For this reason, Shishi prints a warning, suggesting that the user should add the proper 'server-realm' configuration tokens instead.

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.salesservice.asdf.com. 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:

```
_kerberos.foo.asdf.com. IN TXT
```

Finding no match, it would then query:

```
_kerberos.asdf.com. IN TXT
```

handle : Shishi library handle create by **shishi_init()**.

server : hostname to find realm for.

Returns : Returns realm for host, or NULL if not found.

Represent principal name in ASN.1 structure 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 *outlen* does not include the terminating zero.

handle : Shishi library handle create 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 zero terminated string containing principal name. May be **NULL** (to only populate *outlen*).

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

Returns : Returns SHISHI_OK iff 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 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 *outlen* does not include the terminating zero.

handle : Shishi library handle create 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 zero terminated string containing principal name. May be **NULL** (to only populate *outlen*).

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

Returns : Returns SHISHI_OK iff 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 given name.

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 : zero-terminated input array with principal name.

Returns : Returns SHISHI_OK iff successful.

shishi_principal_set ()

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

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

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 : zero-terminated string with principal name on RFC 1964 form.

Returns : Returns SHISHI_OK iff successful.

shishi_parse_name ()

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

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

handle : Shishi library handle create by **shishi_init()**.

name : Input principal name string, e.g. imap/mail.gnu.orgGNU.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 "\", or SHISHI_OK iff 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 principal.

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.extundo.com") based on supplied service name (i.e., "imap") and the system host-name as returned by **hostname()** (i.e., "yxa.extundo.com"). The string must be deallocated by the caller.

handle : shishi handle as allocated by **shishi_init()**.

service : zero 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.

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.

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.

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);
```

handle :

ticket :

realm :

server :

Returns :

shishi_ticket_set_server ()

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

handle :

ticket :

server :

Returns :

shishi_ticket_realm_get ()

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

Extract realm from ticket.

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.

realmrlen : 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.

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.

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.

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.

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);
```

handle :

ticket :

key :

encticketpart :

Returns :

shishi_tkt_ticket ()

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

Get ASN.1 Ticket structure from ticket.

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.

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.

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.

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.

tkt : structure that holds information about Ticket exchange

enckdcreppart : EncKDCRepPart to store in Ticket.

shishi_tkt_enticketpart ()

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

Get ASN.1 EncTicketPart structure from ticket.

tkt : input variable with ticket info.

Returns : Returns EncTicketPart information.

shishi_tkt_enticketpart_set ()

```
void      shishi_tkt_enticketpart_set      (Shishi_tkt *tkt,  
                                             Shishi_asn1 enticketpart);
```

Set the EncTicketPart in the Ticket.

tkt : input variable with ticket info.

enticketpart : enticketpart 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.

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.

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.

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.

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.

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 *realm_len);
```

Extract realm of server in ticket.

tkt : input variable with ticket info.

realm : pointer to newly allocated character array with realm name.

realm_len : 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 *client_len);
```

Represent client principal name in Ticket KDC-REP as zero-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 *client_len* does not include the terminating zero.

tkt : input variable with ticket info.

client : pointer to newly allocated zero terminated string containing principal name. May be **NULL** (to only populate *client_len*).

client_len : 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.

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.

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.

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);
```

tkt :

realm :

client :

Returns :

shishi_tkt_serverrealm_set ()

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

tkt :

realm :

server :

Returns :

shishi_tkt_build ()

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

tkt :

key :

Returns :

shishi_tkt_lastreq ()

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

tkt :

lrtime :

lrtimelen :

lrtype :

Returns :

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.

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).

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);
```

tkt :

authtime :

authtimelen :

Returns :

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.

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);
```

tkt :

starttime :

starttimelen :

Returns :

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).

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);
```

tkt :***endtime*** :***endtimelen*** :**Returns** :**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.

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);
```

tkt :***renewtilltime*** :***renewtilllen*** :**Returns** :**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.

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).

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).

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).

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 allocated 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.

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.

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.

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.

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).

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);
```

tkt :

key :

Returns :

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.

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).

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.

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.

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.

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.

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.

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.

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 be obtained by requesting the postdating in the KRB_AS_REQ message. The life (endtime-starttime) of a postdated ticket will be the remaining life of the 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.

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.

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).

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.

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.

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).

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).

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.

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.

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.

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.

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.

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.

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);
```

tkts :

Returns :

shishi_tkts ()

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

Get a new ticket set handle.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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);
```

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\(\)](#).

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_clientserver\(\)](#) and [shishi_tkts_find\(\)](#).

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 the ticket, then tries to find a suitable TGT, possibly via an AS exchange, using [shishi_tkts_get_tgt\(\)](#), and then use that TGT in a TGS exchange to get the ticket.

Currently this function do not implement cross realm logic.

tkts : ticket set handle as allocated by [shishi_tkts\(\)](#).

hint : structure with characteristics of ticket to begot.

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.

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.

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\(\)](#).

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 for the given server and the default principal client. See [shishi_tkts_get\(\)](#).

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);
```

tkts :

service :

passwd :

Returns :

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\(\)](#).

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.

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.

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.

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.

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.

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.

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);
```

handle :

fh :

enckdcreppart :

Returns :

shishi_enckdcreppart_save ()

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

handle :

fh :

enckdcreppart :

Returns :

shishi_enckdcreppart_parse ()

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

handle :

fh :

enckdcreppart :

Returns :

shishi_enckdcreppart_read ()

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

handle :

fh :

enckdcreppart :

Returns :

shishi_ticket_save ()

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

handle :

fh :

ticket :

Returns :

shishi_ticket_print ()

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

handle :

fh :

ticket :

Returns :

shishi_kdc_print ()

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

handle :

fh :

asreq :

asrep :

encasreppart :

Returns :

shishi_ticket_parse ()

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

handle :

fh :

ticket :

Returns :

shishi_ticket_read ()

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

handle :

fh :

ticket :

Returns :

shishi_etype_info_print ()

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

handle :

fh :

etypeinfo :

Returns :

shishi_etype_info2_print ()

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

handle :

fh :

etypeinfo2 :

Returns :

shishi_padata_print ()

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

handle :

fh :

padata :

Returns :

shishi_methoddata_print ()

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

handle :

fh :

methoddata :

Returns :

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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);
```

handle :

authenticator :

Returns :

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.

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.

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.

handle : shishi handle as allocated by **shishi_init()**.

authenticator : authenticator as allocated by **shishi_authenticator()**.

key : key 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 occurred.

shishi_as_req_build ()

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

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

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.

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.

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.

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.

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 occurred.

shishi_as_rep_set ()

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

Set the AS-REP in the AS exchange.

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.

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.

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.

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.

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 occurred.

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.

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.

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.

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 occurred.

shishi_as_tkt_set ()

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

Set the Tkt in the AS exchange.

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.

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.

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.

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.

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.

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.

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 occurred.

shishi_tgs_tgtkt_set ()

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

Set the Ticket in the TGS exchange.

tgs : structure that holds information about TGS exchange

tgkt : ticket granting ticket to store in TGS.

shishi_tgs_ap ()

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

Get the AP from TGS exchange.

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 occurred.

shishi_tgs_req ()

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

Get the TGS-REQ from TGS exchange.

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 occurred.

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.

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.

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.

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.

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.

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.

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 occurred.

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.

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.

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.

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.

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 occurred.

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.

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.

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.

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 occurred.

shishi_tgs_tkt_set ()

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

Set the Ticket in the TGS exchange.

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.

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.

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.

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.

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.

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);
```

handle :

realm :

service :

req :

Returns :

shishi_asreq ()

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

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

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);
```

handle :

realm :

server :

client :

Returns :

shishi_tgsreq ()

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

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

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);
```

handle :

realm :

server :

tkt :

Returns :

shishi_kdcreq_save ()

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

Print DER encoding of KDC-REQ to file.

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.

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.

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.

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.

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.

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.

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);
```

handle :

kdcreq :

nonce :

Returns :

shishi_kdcreq_nonce_set ()

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

Store nonce number field in KDC-REQ.

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.

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.

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.

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.

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 *realmrlen);
```

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 *realmrlen* does not include the terminating zero.

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 *realmrlen*).

realmrlen : pointer to length of *realm* on output, excluding terminating zero. May be **NULL** (to only populate *realmrlen*).

Returns : Returns SHISHI_OK iff successful.

shishi_kdcreq_realm_get ()

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

handle :

kdcreq :

realm :

realmrlen :

Returns :

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.

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);
```

handle :

req :

service :

Returns :

shishi_kdcreq_set_realmserver ()

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

handle :

req :

realm :

service :

Returns :

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 zero-terminated string. The string is typically 15 characters long. The string is allocated by this function, and it is the responsibility of the caller to deallocate it. Note that the output length *realm*len does not include the terminating zero.

handle : Shishi library handle create by **shishi_init()**.

kdcreq : KDC-REQ variable to get client name from.

till : pointer to newly allocated zero terminated string containing "till" field with generalized time. May be **NULL** (to only populate *realm*len).

tilllen : pointer to length of *till* on output, excluding terminating zero. May be **NULL** (to only populate *tilllen*).

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.

handle : Shishi library handle create by **shishi_init()**.

kdcreq : KDC-REQ variable to get till field from.

Returns : Returns 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.

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.

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.

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.

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.

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 PROXIBLE 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.

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 PROXIBLE 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.

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.

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.

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.

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 transited field must be checked locally. KDCs are encouraged but not required to honor the DISABLE-TRANSITED-CHECK option.

This flag is new since RFC 1510

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.

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.

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.

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.

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.

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.

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.

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.

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).

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.)

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 aprereq);
```

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.

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.

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);
```

handle :

kdcreq :

Returns :

shishi_as_derive_salt ()

```
int                shishi_as_derive_salt            (Shishi *handle,
                                                    Shishi_asn1 asreq,
                                                    Shishi_asn1 asrep,
                                                    char **salt,
                                                    size_t *saltlen);
```

Derive 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 and returns it if found, otherwise the salt is derived from the client name and realm in AS-REQ.

handle : shishi handle as allocated by [shishi_init\(\)](#).

asreq : input AS-REQ variable.

asrep : input AS-REP variable.

salt : newly allocated output array with salt.

saltlen : holds actual size of output array with salt.

Returns : Returns SHISHI_OK iff successful.

shishi_tgs_process ()

```
int                shishi_tgs_process                (Shishi *handle,
                                                    Shishi_asn1 tgsreq,
                                                    Shishi_asn1 tgsrep,
                                                    Shishi_asn1 authenticator,
                                                    Shishi_asn1 oldenckdcreppart,
                                                    Shishi_asn1 *enckdcreppart);
```

Process a TGS client exchange and output decrypted EncKDCRepPart which holds details for the new ticket received. This function simply derives the encryption key from the ticket used to construct the TGS request and calls [shishi_kdc_process\(\)](#), which see.

handle : shishi handle as allocated by [shishi_init\(\)](#).

tgsreq : input variable that holds the sent KDC-REQ.

tgsrep : input variable that holds the received KDC-REP.

authenticator : input variable with Authenticator from AP-REQ in KDC-REQ.

oldenckdcreppart : input variable with EncKDCRepPart used in request.

enckdcreppart : output variable that holds new EncKDCRepPart.

Returns : Returns SHISHI_OK iff the TGS client exchange was successful.

shishi_as_process ()

```
int                shishi_as_process                (Shishi *handle,
                                                    Shishi_asn1 asreq,
                                                    Shishi_asn1 asrep,
                                                    const char *string,
                                                    Shishi_asn1 *enckdcreppart);
```

Process an AS client exchange and output decrypted EncKDCRepPart which holds details for the new ticket received. This function simply derives the encryption key from the password and calls [shishi_kdc_process\(\)](#), which see.

handle : shishi handle as allocated by [shishi_init\(\)](#).

asreq : input variable that holds the sent KDC-REQ.

asrep : input variable that holds the received KDC-REP.

string : input variable with zero terminated password.

enckdcreppart : output variable that holds new EncKDCRepPart.

Returns : Returns SHISHI_OK iff the AS client exchange was successful.

shishi_kdc_process ()

```
int                shishi_kdc_process                (Shishi *handle,
                                                    Shishi_asn1 kdcreq,
                                                    Shishi_asn1 kdcrep,
                                                    Shishi_key *key,
                                                    int keyusage,
                                                    Shishi_asn1 *enckdcreppart);
```

Process a KDC client exchange and output decrypted EncKDCRepPart which holds details for the new ticket received. Use [shishi_kdcrep_get_ticket\(\)](#) to extract the ticket. This function verifies the various conditions that must hold if the response is to be considered valid, specifically it compares nonces ([shishi_kdc_check_nonce\(\)](#)) and if the exchange was a AS exchange, it also compares cname and crealm ([shishi_as_check_cname\(\)](#) and [shishi_as_check_crealm\(\)](#)).

Usually the [shishi_as_process\(\)](#) and [shishi_tgs_process\(\)](#) functions should be used instead, since they simplify the decryption key computation.

handle : shishi handle as allocated by [shishi_init\(\)](#).

kdcreq : input variable that holds the sent KDC-REQ.

kdcrep : input variable that holds the received KDC-REP.

key : input array with key to decrypt encrypted part of KDC-REP with.

keyusage : kerberos key usage value.

enckdcreppart : output variable that holds new EncKDCRepPart.

Returns : Returns SHISHI_OK iff the KDC client exchange was successful.

shishi_kdcreq_sendrecv ()

```
int                shishi_kdcreq_sendrecv            (Shishi *handle,
                                                    Shishi_asn1 kdcreq,
                                                    Shishi_asn1 *kdcrep);
```

handle :

kdcreq :

kdcrep :

Returns :

shishi_kdcreq_sendrecv_hint ()

```
int                shishi_kdcreq_sendrecv_hint       (Shishi *handle,
                                                    Shishi_asn1 kdcreq,
                                                    Shishi_asn1 *kdcrep,
                                                    Shishi_tkts_hint *hint);
```

handle :

kdcreq :

kdcrep :

hint :

Returns :

shishi_kdc_copy_crealm ()

```
int                shishi_kdc_copy_crealm                (Shishi *handle,
                                                         Shishi_asn1 kdcprep,
                                                         Shishi_asn1 encticketpart);
```

Set crealm in KDC-REP to value in EncTicketPart.

handle : shishi handle as allocated by [shishi_init\(\)](#).

kdcprep : KDC-REP to read crealm from.

encticketpart : EncTicketPart to set crealm in.

Returns : Returns SHISHI_OK if successful.

shishi_as_check_crealm ()

```
int                shishi_as_check_crealm                (Shishi *handle,
                                                         Shishi_asn1 asreq,
                                                         Shishi_asn1 asrep);
```

Verify that AS-REQ.req-body.realm and AS-REP.crealm fields matches. This is one of the steps that has to be performed when processing a AS-REQ and AS-REP exchange, see [shishi_kdc_process\(\)](#).

handle : shishi handle as allocated by [shishi_init\(\)](#).

asreq : AS-REQ to compare realm field in.

asrep : AS-REP to compare realm field in.

Returns : Returns SHISHI_OK if successful, SHISHI_REALM_MISMATCH if the values differ, or an error code.

shishi_kdc_copy_cname ()

```
int                shishi_kdc_copy_cname                (Shishi *handle,
                                                         Shishi_asn1 kdcprep,
                                                         Shishi_asn1 encticketpart);
```

Set cname in KDC-REP to value in EncTicketPart.

handle : shishi handle as allocated by [shishi_init\(\)](#).

kdcprep : KDC-REQ to read cname from.

encticketpart : EncTicketPart to set cname in.

Returns : Returns SHISHI_OK if successful.

shishi_as_check_cname ()

```
int                shishi_as_check_cname                (Shishi *handle,
                                                         Shishi_asn1 asreq,
                                                         Shishi_asn1 asrep);
```

Verify that AS-REQ.req-body.realm and AS-REP.crealm fields matches. This is one of the steps that has to be performed when processing a AS-REQ and AS-REP exchange, see [shishi_kdc_process\(\)](#).

handle : shishi handle as allocated by `shishi_init()`.

asreq : AS-REQ to compare client name field in.

asrep : AS-REP to compare client name field in.

Returns : Returns SHISHI_OK if successful, SHISHI_CNAME_MISMATCH if the values differ, or an error code.

shishi_kdc_copy_nonce ()

```
int                shishi_kdc_copy_nonce                (Shishi *handle,
                                                         Shishi_asn1 kdcreq,
                                                         Shishi_asn1 enckdcprep);
```

Set nonce in EncKDCRepPart to value in KDC-REQ.

handle : shishi handle as allocated by `shishi_init()`.

kdcreq : KDC-REQ to read nonce from.

enckdcprep : EncKDCRepPart to set nonce in.

Returns : Returns SHISHI_OK if successful.

shishi_kdc_check_nonce ()

```
int                shishi_kdc_check_nonce                (Shishi *handle,
                                                         Shishi_asn1 kdcreq,
                                                         Shishi_asn1 enckdcprep);
```

Verify that KDC-REQ.req-body.nonce and EncKDCRepPart.nonce fields matches. This is one of the steps that has to be performed when processing a KDC-REQ and KDC-REP exchange.

handle : shishi handle as allocated by `shishi_init()`.

kdcreq : KDC-REQ to compare nonce field in.

enckdcprep : Encrypted KDC-REP part to compare nonce field in.

Returns : Returns SHISHI_OK if successful, SHISHI_NONCE_LENGTH_MISMATCH if the nonces have different lengths (usually indicates that buggy server truncated nonce to 4 bytes), SHISHI_NONCE_MISMATCH if the values differ, or an error code.

shishi_asrep ()

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

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 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.

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.

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.

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 server name field in the KDC-REP.

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.

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);
```

handle :

kdcrep :

crealm :

client :

Returns :

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.

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);
```

handle :

kdcrep :

key :

keyusage :

enckdcreppart :

Returns :

shishi_enckdcreppart ()

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

handle :

Returns :

shishi_encasreppart ()

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

handle :

Returns :

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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);
```

handle :

enckdcreppart :

server :

Returns :

shishi_enckdcreppart_srealmserver_set ()

```
int                shishi_enckdcreppart_srealmserver_set (Shishi *handle,
                                                            Shishi_asn1 enckdcreppart,
                                                            const char *srealm,
                                                            const char *server);
```

handle :

enckdcreppart :

srealm :

server :

Returns :

shishi_enckdcreppart_populate_onticketpart ()

```
int shishi_enckdcreppart_populate_onticketpart
                                     (Shishi *handle,
                                      Shishi_asn1 enckdcreppart,
                                      Shishi_asn1 onticketpart);
```

Set the flags, authtime, starttime, endtime, renew-till and caddr fields of the EncKDCRepPart to the corresponding values in the EncTicketPart.

handle : shishi handle as allocated by **shishi_init()**.

enckdcreppart : input EncKDCRepPart variable.

onticketpart : 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.

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.

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.

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.

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.

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.

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.

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

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.

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 *realmrlen);
```

Extract client realm from KRB-ERROR.

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.

realmrlen : 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.

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.

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 *clientrlen);
```

Return client principal name in KRB-ERROR.

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.

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.

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.

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.

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 *realmrlen);
```

Extract (server) realm from KRB-ERROR.

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.

realmrlen : 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.

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 *serverrlen);
```

Return server principal name in KRB-ERROR.

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 *serverrlen*).

serverrlen : 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.)

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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).

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.

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.

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).

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.

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.

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.

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.

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.

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.

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\(\)](#)).

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](#).

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);
```

Convert C time to KerberosTime. The string must not be deallocate by caller.

handle : shishi handle as allocated by [shishi_init\(\)](#).

t : C time to convert.

Returns : Return a KerberosTime time string corresponding to C time t.

shishi_generalize_now ()

```
const char *      shishi_generalize_now      (Shishi *handle);
```

Convert current time to KerberosTime. The string must not be deallocate by caller.

handle : shishi handle as allocated by **shishi_init()**.

Returns : Return a KerberosTime time string corresponding to current time.

shishi_generalize_ctime ()

```
time_t           shishi_generalize_ctime     (Shishi *handle,  
                                             const char *t);
```

Convert KerberosTime to C time.

handle : shishi handle as allocated by **shishi_init()**.

t : KerberosTime to convert.

Returns : Returns C time corresponding to KerberosTime t.

shishi_time ()

```
int              shishi_time                (Shishi *handle,  
                                             Shishi_asn1 node,  
                                             const char *field,  
                                             char **t);
```

Extract time from ASN.1 structure.

handle : shishi handle as allocated by **shishi_init()**.

node : ASN.1 node to get time from.

field : Name of field in ASN.1 node to get time from.

t : newly allocated output array with zero terminated time string.

Returns : Returns SHISHI_OK iff successful.

shishi_ctime ()

```
int              shishi_ctime               (Shishi *handle,  
                                             Shishi_asn1 node,  
                                             const char *field,  
                                             time_t *t);
```

Extract time from ASN.1 structure.

handle : shishi handle as allocated by **shishi_init()**.

node : ASN.1 variable to read field from.

field : name of field in *node* to read.

t : pointer to time field to set.

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_randomize ()

```
int                shishi_randomize                (Shishi *handle,  
                                                    int strong,  
                                                    void *data,  
                                                    size_t datalen);
```

Store cryptographically random data of given size in the provided buffer.

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 1510. 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

handle: shishi handle as allocated by `shishi_init()`.

key : key to encrypt with.

keyusage: integer specifying what this key is encrypting.

\mathbf{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.

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.

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.

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.

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.

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.

handle: shishi handle as allocated by `shishi_init()`.

key : key to decrypt with.

keyusage: integer specifying what this key is decrypting.

\mathbf{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.

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.

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.

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.

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: $DK(KEY, PRFCONSTANT) = SHISHI_RANDOM-TO-KEY(SHISHI_DR(KEY, PRFCONSTANT))$.

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: $DR(KEY, PRFCONSTANT) = TRUNCATE(DERIVEDRANDOMLEN, SHISHI_ENCRYPT(KEY, PRFCONSTANT))$.

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.com/~shs/papers/betterkey.pdf>, although the sample vectors provided by the paper are incorrect.

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.)

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_decrypt()` 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()`.

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.

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()`.

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()`.

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);
```

Check that the version of the library is at minimum the one given as a string in `req_version`.

req_version : version string to compare with, or NULL

Returns : the actual version string of the library; NULL if the condition is not met. If `NULL` is passed to this function no check is done and only the version string is returned.

shishi_prompt_password_func ()

```
int               (*shishi_prompt_password_func) (Shishi *handle,  
                                                  char **s,  
                                                  const char *format,  
                                                  va_list ap);
```

handle :

s :

format :

ap :

Returns :

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.

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()**.

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.

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);
```

handle :

node :

field :

n :

Returns :

shishi_asn1_empty_p ()

```
int                          shishi_asn1_empty_p          (Shishi *handle,
                                                             Shishi_asn1 node,
                                                             const char *field);
```

handle :

node :

field :

Returns :

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.

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, or 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.

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, or SHISHI_ASN1_ERROR otherwise.

shishi_asn1_read_integer ()

```
int                shishi_asn1_read_integer        (Shishi *handle,
                                                    Shishi_asn1 node,
                                                    const char *field,
                                                    int *i);
```

handle :

node :

field :

i :

Returns :

shishi_asn1_read_int32 ()

```
int                shishi_asn1_read_int32                (Shishi *handle,
                                                         Shishi_asn1 node,
                                                         const char *field,
                                                         int32_t *i);
```

handle :

node :

field :

i :

Returns :

shishi_asn1_read_uint32 ()

```
int                shishi_asn1_read_uint32              (Shishi *handle,
                                                         Shishi_asn1 node,
                                                         const char *field,
                                                         uint32_t *i);
```

handle :

node :

field :

i :

Returns :

shishi_asn1_read_bitstring ()

```
int                shishi_asn1_read_bitstring           (Shishi *handle,
                                                         Shishi_asn1 node,
                                                         const char *field,
                                                         uint32_t *flags);
```

handle :

node :

field :

flags :

Returns :

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.

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);
```

handle :

node :

field :

data :

datalen :

Returns :

shishi_asn1_write_integer ()

```
int                shishi_asn1_write_integer     (Shishi *handle,
                                                    Shishi_asn1 node,
                                                    const char *field,
                                                    int n);
```

handle :

node :

field :

n :

Returns :

shishi_asn1_write_int32 ()

```
int                shishi_asn1_write_int32      (Shishi *handle,
                                                Shishi_asn1 node,
                                                const char *field,
                                                int32_t n);
```

handle :**node :****field :****n :****Returns :****shishi_asn1_write_uint32 ()**

```
int                shishi_asn1_write_uint32    (Shishi *handle,
                                                Shishi_asn1 node,
                                                const char *field,
                                                uint32_t n);
```

handle :**node :****field :****n :****Returns :****shishi_asn1_write_bitstring ()**

```
int                shishi_asn1_write_bitstring (Shishi *handle,
                                                Shishi_asn1 node,
                                                const char *field,
                                                uint32_t flags);
```

handle :**node :****field :****flags :****Returns :****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.

handle : shishi handle as allocated by [shishi_init\(\)](#).

node : ASN.1 node to dellocate.

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

handle : shishi handle as allocated by **shishi_init()**.

Returns : Returns ASN.1 structure.

shishi_asn1_enticketpart ()

```
Shishi_asn1          shishi_asn1_enticketpart          (Shishi *handle);
```

Create new ASN.1 structure for EncTicketPart.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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_krb-safe ()

Shishi_asn1	shishi_der2asn1_krb-safe	(Shishi *handle, const char *der, size_t derlen);
-------------	--------------------------	---

Decode DER encoding of KRB-SAFE and create a ASN.1 structure.

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.

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.

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_entsgsreppart ()

Shishi_asn1	shishi_der2asn1_entsgsreppart	(Shishi *handle, const char *der, size_t derlen);
-------------	-------------------------------	---

Decode DER encoding of EncTGSRepPart and create a ASN.1 structure.

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.

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.

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.

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.

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.

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.

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.

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\(\)](#)).

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.

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.

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.

ap : structure that holds information about AP exchange

tk : 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_tkoptionsdata ()

```
int                shishi_ap_tkoptionsdata      (Shishi *handle,
                                                Shishi_ap **ap,
                                                Shishi_tkt *tk,
                                                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_tkoptionsdata\(\)](#). A random session key is added to the authenticator, using the same keytype as the ticket.

handle : shishi handle as allocated by [shishi_init\(\)](#).

ap : pointer to new structure that holds information about AP exchange

tk : 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_tkoptionsraw ()

```
int                shishi_ap_set_tkoptionsraw   (Shishi_ap *ap,
                                                Shishi_tkt *tk,
                                                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.

ap : structure that holds information about AP exchange

tk : 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.

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.

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.

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.

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 occurred.

shishi_ap_tkt_set ()

```
void                shishi_ap_tkt_set                (Shishi_ap *ap,
                                                       Shishi_tkt *tkt);
```

Set the Ticket in the AP exchange.

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.

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 checksummed, and the checksum placed in the checksum field. It is not the actual checksum field. See also `shishi_ap_authenticator_cksumraw_set`.

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`.

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.

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.

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.

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 occurred.

shishi_ap_authenticator_set ()

```
void              shishi_ap_authenticator_set        (Shishi_ap *ap,  
                                                      Shishi_asn1 authenticator);
```

Set the Authenticator in the AP exchange.

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.

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 occurred.

shishi_ap_req_set ()

```
void                shishi_ap_req_set                (Shishi_ap *ap,  
                                                    Shishi_asn1 apreq);
```

Set the AP-REQ in the AP exchange.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 occurred.

shishi_ap_rep_set ()

```
void                shishi_ap_rep_set                (Shishi_ap *ap,  
                                                       Shishi_asn1 aprepare);
```

Set the AP-REP in the AP exchange.

ap : structure that holds information about AP exchange

aprep : aprepare 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.

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.

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.

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\(\)](#).

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\(\)](#).

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.

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.

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.

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 occurred.

shishi_ap_encapreppart_set ()

```
void                shishi_ap_encapreppart_set        (Shishi_ap *ap,  
                                                    Shishi_asn1 encapreppart);
```

Set the EncAPRepPart in the AP exchange.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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).

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.

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.

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.

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-permanent keys.

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.

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.

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.

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.

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.

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);
```

Print an ASCII representation of a key structure to file descriptor. Example output:

-----BEGIN SHISHI KEY----- Keytype: 18 (aes256-cts-hmac-sha1-96) Principal: host/latte.josefsson.org Realm: JOSEFS-SON.ORG Key-Version-Number: 1

P1QdeW/oSiag/bTyVEBAY2msiGSTmgLXlopuCKoppDs= -----END SHISHI KEY-----

handle : shishi handle as allocated by **shishi_init()**.

fh : file handle opened for writing.

key : key to print.

Returns : Returns SHISHI_OK iff successful.

shishi_key_to_file ()

```
int                shishi_key_to_file                (Shishi *handle,
                                                    const char *filename,
                                                    Shishi_key *key);
```

Print an ASCII representation of a key structure to a file. The file is appended to if it exists. See [shishi_key_print\(\)](#) for format of output.

handle : shishi handle as allocated by [shishi_init\(\)](#).

filename : filename to append key to.

key : key to print.

Returns : Returns SHISHI_OK iff successful.

shishi_key_parse ()

```
int                shishi_key_parse                (Shishi *handle,
                                                    FILE *fh,
                                                    Shishi_key **key);
```

handle :

fh :

key :

Returns :

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.

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.

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.

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.

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.

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.

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.

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.

keys : key set handle as allocated by **shishi_keys()**.

shishi_keys_size ()

```
int                shishi_keys_size          (Shishi_keys *keys);
```

Get size of key set.

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.

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.

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.

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](#).

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.

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.

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*.

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*.

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*.

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 host-name), 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.

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.

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.

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.

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.

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.

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.

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.

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*.

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*.

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*.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 apreql);
```

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

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.

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.

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.

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.

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.

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.

handle : shishi handle as allocated by [shishi_init\(\)](#).

apreq : AP-REQ to add authenticator field to.

key : key 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.

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.

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.

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.

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.

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.

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.

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.

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);
```

handle :

apreq :

key :

keyusage :

authenticator :

Returns :

shishi_aprep ()

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

This function creates a new AP-REP, populated with some default values.

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.

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.

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.

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.

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.

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.

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);
```

handle :

aprep :

key :

keyusage :

encapreppart :

Returns :

shishi_aprep_verify ()

```
int                shishi_aprep_verify        (Shishi *handle,  
                                                  Shishi_asn1 authenticator,  
                                                  Shishi_asn1 encapreppart);
```

handle :

authenticator :

encapreppart :

Returns :

shishi_aprep_enc_part_set ()

```
int                shishi_aprep_enc_part_set      (Shishi *handle,  
                                                    Shishi_asn1 aprep,  
                                                    int etype,  
                                                    const char *buf,  
                                                    size_t buflen);
```

handle :

aprep :

etype :

buf :

buflen :

Returns :

shishi_aprep_enc_part_add ()

```
int                shishi_aprep_enc_part_add      (Shishi *handle,  
                                                    Shishi_asn1 aprep,  
                                                    Shishi_asn1 encticketpart,  
                                                    Shishi_asn1 encapreppart);
```

handle :

aprep :

encticketpart :

encapreppart :

Returns :

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);
```

handle :

aprep :

encapreppart :

authenticator :

encticketpart :

Returns :

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.

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.

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.

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);
```

handle :

Returns :

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.

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.

handle : shishi handle as allocated by **shishi_init()**.

encticketpart : input EncTicketPart variable.

key : newly allocated key.

Returns : Returns **SHISHI_OK** iff successful.

shishi_ecticketpart_crealm ()

```
int                shishi_ecticketpart_crealm      (Shishi *handle,
                                                    Shishi_asn1  ecticketpart,
                                                    char **crealm,
                                                    size_t *crealmolen);
```

handle :

ecticketpart :

crealm :

crealmolen :

Returns :

shishi_ecticketpart_crealm_set ()

```
int                shishi_ecticketpart_crealm_set  (Shishi *handle,
                                                    Shishi_asn1  ecticketpart,
                                                    const char *realm);
```

Set the realm field in the KDC-REQ.

handle : shishi handle as allocated by **shishi_init()**.

ecticketpart : input EncTicketPart variable.

realm : input array with name of realm.

Returns : Returns SHISHI_OK iff successful.

shishi_ecticketpart_client ()

```
int                shishi_ecticketpart_client      (Shishi *handle,
                                                    Shishi_asn1  ecticketpart,
                                                    char **client,
                                                    size_t *clientolen);
```

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 *clientolen* does not include the terminating zero.

handle : Shishi library handle create by **shishi_init()**.

ecticketpart : EncTicketPart variable to get client name from.

client : pointer to newly allocated zero terminated string containing principal name. May be **NULL** (to only populate *clientolen*).

clientolen : pointer to length of *client* on output, excluding terminating zero. May be **NULL** (to only populate *client*).

Returns : Returns SHISHI_OK iff successful.

shishi_encyticketpart_clientrealm ()

```
int                shishi_encyticketpart_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 allocated 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.

handle : Shishi library handle created by **shishi_init()**.

encyticketpart : 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_encyticketpart_cname_set ()

```
int                shishi_encyticketpart_cname_set    (Shishi *handle,
                                                         Shishi_asn1 encticketpart,
                                                         Shishi_name_type name_type,
                                                         const char *principal);
```

Set the client name field in the EncTicketPart.

handle : shishi handle as allocated by **shishi_init()**.

encyticketpart : 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_encyticketpart_print ()

```
int                shishi_encyticketpart_print        (Shishi *handle,
                                                         FILE *fh,
                                                         Shishi_asn1 encticketpart);
```

handle :

fh :

encyticketpart :

Returns :

shishi_encticketpart_flags_set ()

```
int                shishi_encticketpart_flags_set    (Shishi *handle,
                                                    Shishi_asn1 enticketpart,
                                                    int flags);
```

Set the EncTicketPart.flags to supplied value.

handle : shishi handle as allocated by **shishi_init()**.

enticketpart : input EncTicketPart variable.

flags : flags to set in enticketpart.

Returns : Returns **SHISHI_OK** iff successful.

shishi_encticketpart_transited_set ()

```
int                shishi_encticketpart_transited_set (Shishi *handle,
                                                    Shishi_asn1 enticketpart,
                                                    int32_t trtype,
                                                    const char *trdata,
                                                    size_t trdatalen);
```

Set the EncTicketPart.transited field to supplied value.

handle : shishi handle as allocated by **shishi_init()**.

enticketpart : 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 enticketpart,
                                                    const char *authtime);
```

Set the EncTicketPart.authtime to supplied value.

handle : shishi handle as allocated by **shishi_init()**.

enticketpart : input EncTicketPart variable.

authtime : character buffer containing a generalized time string.

Returns : Returns **SHISHI_OK** iff successful.

shishi_entsicketpart_endtime_set ()

```
int                shishi_entsicketpart_endtime_set    (Shishi *handle,
                                                         Shishi_asn1 entsicketpart,
                                                         const char *endtime);
```

Set the EncTicketPart.endtime to supplied value.

handle : shishi handle as allocated by **shishi_init()**.

entsicketpart : input EncTicketPart variable.

endtime : character buffer containing a generalized time string.

Returns : Returns **SHISHI_OK** iff successful.

shishi_entsicketpart_authtime ()

```
int                shishi_entsicketpart_authtime      (Shishi *handle,
                                                         Shishi_asn1 entsicketpart,
                                                         char *authtime,
                                                         size_t *authtimelen);
```

handle :

entsicketpart :

authtime :

authtimelen :

Returns :

shishi_entsicketpart_authctime ()

```
time_t             shishi_entsicketpart_authctime     (Shishi *handle,
                                                         Shishi_asn1 entsicketpart);
```

handle :

entsicketpart :

Returns :

shishi_safe ()

```
int                shishi_safe                        (Shishi *handle,
                                                         Shishi_safe **safe);
```

Create a new SAFE exchange.

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.

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.

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 occurred.

shishi_safe_key_set ()

```
void                shishi_safe_key_set            (Shishi_safe *safe,  
                                                    Shishi_key *key);
```

Set the Key in the SAFE exchange.

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.

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 occurred.

shishi_safe_safe_set ()

```
void                shishi_safe_safe_set          (Shishi_safe *safe,  
                                                    Shishi_asn1 asn1safe);
```

Set the KRB-SAFE in the SAFE exchange.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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_ if the actual verification failed.

shishi_priv ()

```
int                shishi_priv                (Shishi *handle,  
                                                    Shishi_priv **priv);
```

Create a new PRIV exchange.

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.

priv : structure that holds information about PRIV exchange

shishi_priv_key ()

```
Shishi_key *        shishi_priv_key                (Shishi_priv *priv);
```

Get key from PRIV exchange.

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 occurred.

shishi_priv_key_set ()

```
void                shishi_priv_key_set                (Shishi_priv *priv,  
                                                    Shishi_key *key);
```

Set the Key in the PRIV exchange.

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.

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 occurred.

shishi_priv_priv_set ()

```
void      shishi_priv_priv_set      (Shishi_priv *priv,  
                                     Shishi_asn1 asn1priv);
```

Set the KRB-PRIV in the PRIV exchange.

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.

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.

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.

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 occurred.

shishi_priv_encprivpart_set ()

```
void      shishi_priv_encprivpart_set      (Shishi_priv *priv,  
                                             Shishi_asn1 asnlencprivpart);
```

Set the ENCPRIVPART in the PRIV exchange.

priv : structure that holds information about PRIV exchange

asnlencprivpart : 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.

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.

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.

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.

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.

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.

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.

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.

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.

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 encrypts the data and calls this function in one step.

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.

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.

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.

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.

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_CRYPT 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. Currently this function only compare the principal name with *authzname* using **strcmp()**.

handle : shishi handle as allocated by **shishi_init()**.

tkt : input variable with ticket info.

authzname : authorization name.

Returns : Returns 1 if authzname is authorized for services by authenticated client principal, or 0 otherwise.

shishi_authorization_parse ()

```
int                shishi_authorization_parse      (const char *authorization);
```

Parse authorization type name.

authorization : name of authorization type, e.g. "basic".

Returns : Return authorization type corresponding to a string.

shishi_authorize_strcmp ()

```
int                shishi_authorize_strcmp        (Shishi *handle,  
                                                  const char *principal,  
                                                  const char *authzname);
```

handle :

principal :

authzname :

Returns :

shishi_authorize_k5login ()

```
int                shishi_authorize_k5login       (Shishi *handle,  
                                                  const char *principal,  
                                                  const char *authzname);
```

handle :

principal :

authzname :

Returns :

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.

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.

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.

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.

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.

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.

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.

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.

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.

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);
```

p :

now :

Returns :

shishi_xalloc_die ()

```
void                shishi_xalloc_die                (void);
```

shishi_resolv ()

```
Shishi_dns          shishi_resolv                    (const char *zone,
                                                         uint16_t querytype);
```

Query DNS resolver for data of type *querytype* at owner name *zone*. Currently TXT and SRV types are supported.

zone : owner name of data, e.g. "EXAMPLE.ORG"

querytype : type of data to query for, e.g., SHISHI_DNS_TXT.

Returns : Returns linked list of DNS records, or NULL if query failed.

shishi_resolv_free ()

```
void                shishi_resolv_free                (Shishi_dns rrs);
```

Deallocate list of DNS RR as returned by **shishi_resolv()**.

rrs : list of DNS RR as returned by **shishi_resolv()**.

Chapter 2

Index

S

- Shishi, 50
- shishi, 51
- shishi_3des, 157
- shishi_aes_cts, 158
- shishi_alloc_fail_function, 51
- Shishi_ap, 50
- shishi_ap, 191
- shishi_ap_authenticator, 198
- shishi_ap_authenticator_cksumdata, 197
- shishi_ap_authenticator_cksumdata_set, 197
- shishi_ap_authenticator_cksumraw_set, 197
- shishi_ap_authenticator_cksumtype, 198
- shishi_ap_authenticator_cksumtype_set, 198
- shishi_ap_authenticator_set, 198
- shishi_ap_done, 192
- shishi_ap_encapreppart, 203
- shishi_ap_encapreppart_set, 203
- shishi_ap_etype, 192
- shishi_ap_etype_tktoptionsdata, 193
- shishi_ap_key, 200
- shishi_ap_nosubkey, 192
- shishi_ap_option2string, 203
- shishi_ap_rep, 201
- shishi_ap_rep_asn1, 203
- shishi_ap_rep_build, 203
- shishi_ap_rep_der, 201
- shishi_ap_rep_der_set, 202
- shishi_ap_rep_set, 201
- shishi_ap_rep_verify, 202
- shishi_ap_rep_verify_asn1, 202
- shishi_ap_rep_verify_der, 202
- shishi_ap_req, 198
- shishi_ap_req_asn1, 200
- shishi_ap_req_build, 199
- shishi_ap_req_decode, 200
- shishi_ap_req_der, 199
- shishi_ap_req_der_set, 199
- shishi_ap_req_process, 200
- shishi_ap_req_process_keyusage, 201
- shishi_ap_req_set, 199
- shishi_ap_set_tktoptions, 192
- shishi_ap_set_tktoptionsasn1usage, 195
- shishi_ap_set_tktoptionsdata, 193
- shishi_ap_set_tktoptionsraw, 194
- shishi_ap_string2option, 204
- shishi_ap_tkt, 196
- shishi_ap_tkt_set, 196
- shishi_ap_tktoptions, 193
- shishi_ap_tktoptionsasn1usage, 196
- shishi_ap_tktoptionsdata, 194
- shishi_ap_tktoptionsraw, 195
- Shishi_apoptions, 42
- shishi_aprep, 227
- shishi_aprep_decrypt, 229
- shishi_aprep_enc_part_add, 230
- shishi_aprep_enc_part_make, 230
- shishi_aprep_enc_part_set, 230
- shishi_aprep_from_file, 229
- shishi_aprep_get_enc_part_etype, 231
- shishi_aprep_parse, 228
- shishi_aprep_print, 227
- shishi_aprep_read, 228
- shishi_aprep_save, 227
- shishi_aprep_to_file, 228
- shishi_aprep_verify, 229
- shishi_apreq, 222
- shishi_apreq_add_authenticator, 224
- shishi_apreq_decrypt, 227
- shishi_apreq_from_file, 222
- shishi_apreq_get_authenticator_etype, 226
- shishi_apreq_get_ticket, 226
- shishi_apreq_mutual_required_p, 225
- shishi_apreq_options, 225
- shishi_apreq_options_add, 226
- shishi_apreq_options_remove, 226
- shishi_apreq_options_set, 225
- shishi_apreq_parse, 222
- shishi_apreq_print, 222
- shishi_apreq_read, 223
- shishi_apreq_save, 223
- shishi_apreq_set_authenticator, 224
- shishi_apreq_set_ticket, 224
- shishi_apreq_to_file, 223
- shishi_apreq_use_session_key_p, 225
- shishi_arcfour, 156

Shishi_as, 50
shishi_as, 102
shishi_as_check_cname, 129
shishi_as_check_crealm, 129
shishi_as_derive_salt, 126
shishi_as_done, 102
shishi_as_krberror, 105
shishi_as_krberror_der, 105
shishi_as_krberror_set, 105
shishi_as_process, 127
shishi_as_rep, 103
shishi_as_rep_build, 104
shishi_as_rep_der, 104
shishi_as_rep_der_set, 104
shishi_as_rep_process, 106
shishi_as_rep_set, 104
shishi_as_req, 102
shishi_as_req_build, 102
shishi_as_req_der, 103
shishi_as_req_der_set, 103
shishi_as_req_set, 103
shishi_as_sendrecv, 106
shishi_as_sendrecv_hint, 106
shishi_as_tkt, 105
shishi_as_tkt_set, 105
Shishi_asn1, 51
shishi_asn1_aprep, 182
shishi_asn1_apreq, 182
shishi_asn1_asrep, 181
shishi_asn1_asreq, 181
shishi_asn1_authenticator, 183
shishi_asn1_done, 179
shishi_asn1_empty_p, 175
shishi_asn1_encapreppart, 182
shishi_asn1_encasreppart, 183
shishi_asn1_enckdcreppart, 183
shishi_asn1_encprivpart, 184
shishi_asn1_encrypteddata, 180
shishi_asn1_enticketpart, 182
shishi_asn1_etype_info, 180
shishi_asn1_etype_info2, 181
shishi_asn1_krberror, 183
shishi_asn1_krbsafe, 183
shishi_asn1_methoddata, 180
shishi_asn1_msgtype, 185
shishi_asn1_number_of_elements, 175
shishi_asn1_pa_enc_ts_enc, 180
shishi_asn1_padata, 180
shishi_asn1_print, 185
shishi_asn1_priv, 184
shishi_asn1_read, 176
shishi_asn1_read_bitstring, 177
shishi_asn1_read_inline, 176
shishi_asn1_read_int32, 177
shishi_asn1_read_integer, 176
shishi_asn1_read_optional, 178
shishi_asn1_read_uint32, 177
shishi_asn1_tgsrep, 181
shishi_asn1_tgsreq, 181
shishi_asn1_ticket, 182
shishi_asn1_to_der, 184
shishi_asn1_to_der_field, 184
shishi_asn1_write, 178
shishi_asn1_write_bitstring, 179
shishi_asn1_write_int32, 179
shishi_asn1_write_integer, 178
shishi_asn1_write_uint32, 179
shishi_asrep, 130
shishi_asreq, 112
shishi_asreq_clientrealm, 115
shishi_asreq_rsc, 112
shishi_authenticator, 91
shishi_authenticator_add_authorizationdata, 99
shishi_authenticator_add_cksum, 96
shishi_authenticator_add_cksum_type, 96
shishi_authenticator_add_random_subkey, 98
shishi_authenticator_add_random_subkey_etype, 98
shishi_authenticator_add_subkey, 99
shishi_authenticator_authorizationdata, 100
shishi_authenticator_cksum, 95
shishi_authenticator_clear_authorizationdata, 99
shishi_authenticator_client, 94
shishi_authenticator_client_set, 92
shishi_authenticator_clientrealm, 95
shishi_authenticator_ctime, 92
shishi_authenticator_ctime_set, 93
shishi_authenticator_cusec_get, 93
shishi_authenticator_cusec_set, 93
shishi_authenticator_from_file, 101
shishi_authenticator_get_subkey, 97
shishi_authenticator_parse, 100
shishi_authenticator_print, 101
shishi_authenticator_read, 100
shishi_authenticator_remove_cksum, 95
shishi_authenticator_remove_subkey, 97
shishi_authenticator_save, 102
shishi_authenticator_seqnumber_get, 93
shishi_authenticator_seqnumber_remove, 94
shishi_authenticator_seqnumber_set, 94
shishi_authenticator_set_cksum, 96
shishi_authenticator_set_cname, 92
shishi_authenticator_set_crealm, 91
shishi_authenticator_set_subkey, 98
shishi_authenticator_subkey, 97
shishi_authenticator_to_file, 101
Shishi_authorization, 45
shishi_authorization_parse, 249
shishi_authorize_k5login, 249
shishi_authorize_strcmp, 249
shishi_authorized_p, 248
shishi_cfg, 53
shishi_cfg_authorizationtype_set, 55
shishi_cfg_clientkdcetype, 54
shishi_cfg_clientkdcetype_fast, 55

shishi_cfg_clientkdcetype_set, 55
shishi_cfg_default_systemfile, 54
shishi_cfg_default_userdirectory, 54
shishi_cfg_default_userfile, 54
shishi_cfg_from_file, 53
shishi_cfg_print, 53
shishi_cfg_userdirectory_file, 54
shishi_check_version, 174
shishi_checksum, 170
shishi_checksum_cksumlen, 160
shishi_checksum_name, 160
shishi_checksum_parse, 160
shishi_checksum_supported_p, 160
shishi_cipher_blocksize, 159
shishi_cipher_confoundersize, 159
shishi_cipher_defaultcksumtype, 159
shishi_cipher_keylen, 159
shishi_cipher_name, 158
shishi_cipher_parse, 160
shishi_cipher_randomlen, 159
shishi_cipher_supported_p, 158
Shishi_cksumtype, 44
shishi_crc, 154
Shishi_crypto, 51
shishi_crypto, 172
shishi_crypto_close, 173
shishi_crypto_decrypt, 173
shishi_crypto_encrypt, 173
shishi_ctime, 153
shishi_decrypt, 169
shishi_decrypt_etype, 167
shishi_decrypt_iv, 168
shishi_decrypt_iv_etype, 166
shishi_decrypt_ivupdate, 168
shishi_decrypt_ivupdate_etype, 166
shishi_der2asn1, 185
shishi_der2asn1_aprep, 189
shishi_der2asn1_apreq, 189
shishi_der2asn1_asrep, 188
shishi_der2asn1_asreq, 187
shishi_der2asn1_authenticator, 189
shishi_der2asn1_encapreppart, 191
shishi_der2asn1_encasreppart, 190
shishi_der2asn1_enckdcreppart, 191
shishi_der2asn1_encprivpart, 191
shishi_der2asn1_enctgsreppart, 190
shishi_der2asn1_enticketpart, 187
shishi_der2asn1_etype_info, 186
shishi_der2asn1_etype_info2, 186
shishi_der2asn1_kdcrep, 188
shishi_der2asn1_kdcreq, 188
shishi_der2asn1_krberror, 189
shishi_der2asn1_krbsafe, 190
shishi_der2asn1_methoddata, 186
shishi_der2asn1_padata, 186
shishi_der2asn1_priv, 190
shishi_der2asn1_tgsrep, 188
shishi_der2asn1_tgsreq, 187
shishi_der2asn1_ticket, 187
shishi_der_msgtype, 185
shishi_derive_default_salt, 61
shishi_des, 157
shishi_des_cbc_mac, 156
shishi_dk, 171
Shishi_dns, 49
SHISHI_DNS_IN, 49
SHISHI_DNS_SRV, 49
Shishi_dns_srv, 50
SHISHI_DNS_TXT, 49
shishi_done, 51
shishi_dr, 171
shishi_encapreppart, 217
shishi_encapreppart_ctime, 218
shishi_encapreppart_ctime_set, 218
shishi_encapreppart_cusec_get, 218
shishi_encapreppart_cusec_set, 218
shishi_encapreppart_from_file, 220
shishi_encapreppart_get_key, 221
shishi_encapreppart_parse, 220
shishi_encapreppart_print, 219
shishi_encapreppart_read, 220
shishi_encapreppart_save, 219
shishi_encapreppart_seqnumber_get, 221
shishi_encapreppart_seqnumber_remove, 221
shishi_encapreppart_seqnumber_set, 221
shishi_encapreppart_time_copy, 217
shishi_encapreppart_to_file, 219
shishi_encasreppart, 136
shishi_enckdcreppart, 136
shishi_enckdcreppart_authtime_set, 137
shishi_enckdcreppart_endtime_set, 138
shishi_enckdcreppart_flags_set, 137
shishi_enckdcreppart_get_key, 136
shishi_enckdcreppart_key_set, 137
shishi_enckdcreppart_nonce_set, 137
shishi_enckdcreppart_parse, 88
shishi_enckdcreppart_populate_enticketpart, 140
shishi_enckdcreppart_print, 88
shishi_enckdcreppart_read, 89
shishi_enckdcreppart_renew_till_set, 138
shishi_enckdcreppart_save, 88
shishi_enckdcreppart_server_set, 139
shishi_enckdcreppart_sname_set, 139
shishi_enckdcreppart_srealm_set, 138
shishi_enckdcreppart_srealmserver_set, 139
shishi_enckdcreppart_starttime_set, 138
shishi_encprivpart_set_user_data, 248
shishi_encprivpart_user_data, 247
shishi_encrypt, 165
shishi_encrypt_etype, 163
shishi_encrypt_iv, 164
shishi_encrypt_iv_etype, 162
shishi_encrypt_ivupdate, 164
shishi_encrypt_ivupdate_etype, 162

shishi_entticketpart, 232
shishi_entticketpart_authctime, 236
shishi_entticketpart_authtime, 236
shishi_entticketpart_authtime_set, 235
shishi_entticketpart_client, 233
shishi_entticketpart_clientrealm, 234
shishi_entticketpart_cname_set, 234
shishi_entticketpart_crealm, 233
shishi_entticketpart_crealm_set, 233
shishi_entticketpart_endtime_set, 236
shishi_entticketpart_flags_set, 235
shishi_entticketpart_get_key, 232
shishi_entticketpart_key_set, 232
shishi_entticketpart_print, 234
shishi_entticketpart_transited_set, 235
shishi_error, 55
shishi_error_clear, 56
shishi_error_outputtype, 56
shishi_error_printf, 56
shishi_error_set, 56
shishi_error_set_outputtype, 56
Shishi_etype, 44
shishi_etype_info2_print, 90
shishi_etype_info_print, 90
Shishi_filetype, 45
shishi_generalize_ctime, 153
shishi_generalize_now, 153
shishi_generalize_time, 152
SHISHI_GENERALIZEDTIME_LENGTH, 51
SHISHI_GENERALIZEDTIMEZ_LENGTH, 51
shishi_get_date, 251
shishi_hmac_md5, 155
shishi_hmac_sha1, 155
shishi_hostkeys_default_file, 216
shishi_hostkeys_default_file_set, 216
shishi_hostkeys_for_localservice, 217
shishi_hostkeys_for_localservicerealm, 217
shishi_hostkeys_for_server, 216
shishi_hostkeys_for_serverrealm, 216
shishi_info, 57
shishi_init, 52
shishi_init_server, 52
shishi_init_server_with_paths, 53
shishi_init_with_paths, 52
shishi_kdc_check_nonce, 130
shishi_kdc_copy_cname, 129
shishi_kdc_copy_crealm, 129
shishi_kdc_copy_nonce, 130
shishi_kdc_print, 89
shishi_kdc_process, 128
shishi_kdc_sendrecv, 231
shishi_kdc_sendrecv_hint, 231
Shishi_KDCOptions, 43
shishi_kdcrep_add_enc_part, 133
shishi_kdcrep_clear_padata, 132
shishi_kdcrep_client_set, 135
shishi_kdcrep_cname_set, 134
shishi_kdcrep_crealm_set, 134
shishi_kdcrep_crealmserver_set, 135
shishi_kdcrep_decrypt, 136
shishi_kdcrep_from_file, 132
shishi_kdcrep_get_enc_part_etype, 133
shishi_kdcrep_get_ticket, 133
shishi_kdcrep_parse, 132
shishi_kdcrep_print, 131
shishi_kdcrep_read, 132
shishi_kdcrep_save, 131
shishi_kdcrep_set_enc_part, 135
shishi_kdcrep_set_ticket, 134
shishi_kdcrep_to_file, 131
shishi_kdcreq, 112
shishi_kdcreq_add_padata, 125
shishi_kdcreq_add_padata_preauth, 126
shishi_kdcreq_add_padata_tgs, 126
shishi_kdcreq_allow_postdate_p, 121
shishi_kdcreq_build, 126
shishi_kdcreq_clear_padata, 124
shishi_kdcreq_client, 116
shishi_kdcreq_disable_transited_check_p, 122
shishi_kdcreq_enc_tkt_in_skey_p, 123
shishi_kdcreq_etype, 119
shishi_kdcreq_forwardable_p, 120
shishi_kdcreq_forwarded_p, 121
shishi_kdcreq_from_file, 115
shishi_kdcreq_get_padata, 125
shishi_kdcreq_get_padata_tgs, 125
shishi_kdcreq_nonce, 115
shishi_kdcreq_nonce_set, 116
shishi_kdcreq_options, 120
shishi_kdcreq_options_add, 124
shishi_kdcreq_options_set, 124
shishi_kdcreq_parse, 114
shishi_kdcreq_postdated_p, 122
shishi_kdcreq_print, 113
shishi_kdcreq_proxiable_p, 121
shishi_kdcreq_proxy_p, 121
shishi_kdcreq_read, 114
shishi_kdcreq_realm, 117
shishi_kdcreq_realm_get, 118
shishi_kdcreq_renew_p, 123
shishi_kdcreq_renewable_ok_p, 123
shishi_kdcreq_renewable_p, 122
shishi_kdcreq_save, 113
shishi_kdcreq_sendrecv, 128
shishi_kdcreq_sendrecv_hint, 128
shishi_kdcreq_server, 117
shishi_kdcreq_set_cname, 116
shishi_kdcreq_set_etype, 120
shishi_kdcreq_set_realm, 118
shishi_kdcreq_set_realmsrvr, 118
shishi_kdcreq_set_server, 118
shishi_kdcreq_set_sname, 117
shishi_kdcreq_till, 119
shishi_kdcreq_tillc, 119

shishi_kdcreq_to_file, 114
shishi_kdcreq_validate_p, 124
Shishi_key, 50
shishi_key, 207
shishi_key_copy, 207
shishi_key_done, 207
shishi_key_from_base64, 209
shishi_key_from_name, 210
shishi_key_from_random, 209
shishi_key_from_string, 210
shishi_key_from_value, 208
shishi_key_length, 206
shishi_key_name, 205
shishi_key_parse, 208
shishi_key_principal, 204
shishi_key_principal_set, 204
shishi_key_print, 207
shishi_key_random, 208
shishi_key_realm, 204
shishi_key_realm_set, 204
shishi_key_timestamp, 206
shishi_key_timestamp_set, 206
shishi_key_to_file, 208
shishi_key_type, 205
shishi_key_type_set, 205
shishi_key_value, 205
shishi_key_value_set, 205
shishi_key_version, 206
shishi_key_version_set, 206
Shishi_keys, 50
shishi_keys, 211
shishi_keys_add, 212
shishi_keys_add_keytab_file, 214
shishi_keys_add_keytab_mem, 214
shishi_keys_done, 211
shishi_keys_for_localservicerealm_in_file, 213
shishi_keys_for_server_in_file, 213
shishi_keys_for_serverrealm_in_file, 213
shishi_keys_from_file, 212
shishi_keys_from_keytab_file, 215
shishi_keys_from_keytab_mem, 214
shishi_keys_nth, 211
shishi_keys_print, 212
shishi_keys_remove, 211
shishi_keys_size, 211
shishi_keys_to_file, 212
shishi_keys_to_keytab_file, 215
shishi_keys_to_keytab_mem, 215
Shishi_keyusage, 45
Shishi_krb_error, 47
shishi_krberror, 140
shishi_krberror_build, 142
shishi_krberror_client, 143
shishi_krberror_client_set, 144
shishi_krberror_crealm, 143
shishi_krberror_ctime, 146
shishi_krberror_ctime_set, 147
shishi_krberror_cusec, 147
shishi_krberror_cusec_set, 147
shishi_krberror_der, 142
shishi_krberror_edata, 150
shishi_krberror_errorcode, 151
shishi_krberror_errorcode_fast, 151
shishi_krberror_errorcode_message, 152
shishi_krberror_errorcode_set, 149
shishi_krberror_etext, 149
shishi_krberror_from_file, 142
shishi_krberror_message, 152
shishi_krberror_methoddata, 152
shishi_krberror_parse, 141
shishi_krberror_pretty_print, 151
shishi_krberror_print, 140
shishi_krberror_read, 141
shishi_krberror_realm, 145
shishi_krberror_remove_cname, 144
shishi_krberror_remove_crealm, 143
shishi_krberror_remove_ctime, 147
shishi_krberror_remove_cusec, 148
shishi_krberror_remove_edata, 151
shishi_krberror_remove_etext, 150
shishi_krberror_remove_sname, 146
shishi_krberror_save, 140
shishi_krberror_server, 145
shishi_krberror_server_set, 146
shishi_krberror_set_cname, 144
shishi_krberror_set_crealm, 143
shishi_krberror_set_edata, 150
shishi_krberror_set_etext, 150
shishi_krberror_set_realm, 145
shishi_krberror_set_sname, 146
shishi_krberror_stime, 148
shishi_krberror_stime_set, 148
shishi_krberror_susec, 148
shishi_krberror_susec_set, 149
shishi_krberror_to_file, 141
Shishi_lrtype, 44
shishi_md4, 154
shishi_md5, 155
shishi_methoddata_print, 91
Shishi_msgtype, 43
shishi_n_fold, 171
Shishi_name_type, 41
Shishi_outputtype, 45
shishi_padata_print, 91
Shishi_padata_type, 41
shishi_parse_name, 61
shishi_pbkdf2_sha1, 172
shishi_principal_default, 59
shishi_principal_default_guess, 59
shishi_principal_default_set, 59
shishi_principal_name, 59
shishi_principal_name_realm, 60
shishi_principal_name_set, 60
shishi_principal_set, 61

Shishi_priv, 50
shishi_priv, 242
shishi_priv_build, 248
shishi_priv_done, 242
shishi_priv_enc_part_etype, 247
shishi_priv_encprivpart, 244
shishi_priv_encprivpart_der, 244
shishi_priv_encprivpart_der_set, 244
shishi_priv_encprivpart_set, 244
shishi_priv_from_file, 246
shishi_priv_key, 242
shishi_priv_key_set, 242
shishi_priv_parse, 246
shishi_priv_print, 245
shishi_priv_priv, 243
shishi_priv_priv_der, 243
shishi_priv_priv_der_set, 243
shishi_priv_priv_set, 243
shishi_priv_process, 248
shishi_priv_read, 246
shishi_priv_save, 245
shishi_priv_set_enc_part, 247
shishi_priv_to_file, 245
shishi_prompt_password, 175
shishi_prompt_password_callback_get, 175
shishi_prompt_password_callback_set, 174
shishi_prompt_password_func, 174
shishi_random_to_key, 161
shishi_randomize, 154
Shishi_rc, 40
shishi_realm_default, 58
shishi_realm_default_guess, 57
shishi_realm_default_set, 58
shishi_realm_for_server, 59
shishi_realm_for_server_dns, 58
shishi_realm_for_server_file, 58
shishi_resolv, 251
shishi_resolv_free, 251
Shishi_safe, 50
shishi_safe, 236
shishi_safe_build, 241
shishi_safe_cksum, 240
shishi_safe_done, 237
shishi_safe_from_file, 240
shishi_safe_key, 237
shishi_safe_key_set, 237
shishi_safe_parse, 239
shishi_safe_print, 238
shishi_safe_read, 239
shishi_safe_safe, 237
shishi_safe_safe_der, 238
shishi_safe_safe_der_set, 238
shishi_safe_safe_set, 237
shishi_safe_save, 238
shishi_safe_set_cksum, 240
shishi_safe_set_user_data, 241
shishi_safe_to_file, 239
shishi_safe_user_data, 241
shishi_safe_verify, 242
shishi_server, 51
shishi_server_for_local_service, 62
shishi_strerror, 55
shishi_string_to_key, 161
Shishi_tgs, 50
shishi_tgs, 106
shishi_tgs_ap, 107
shishi_tgs_done, 107
shishi_tgs_krberror, 110
shishi_tgs_krberror_der, 110
shishi_tgs_krberror_set, 110
shishi_tgs_process, 127
shishi_tgs_rep, 109
shishi_tgs_rep_build, 109
shishi_tgs_rep_der, 109
shishi_tgs_rep_process, 110
shishi_tgs_req, 107
shishi_tgs_req_build, 108
shishi_tgs_req_der, 108
shishi_tgs_req_der_set, 108
shishi_tgs_req_process, 109
shishi_tgs_req_set, 108
shishi_tgs_sendrecv, 111
shishi_tgs_sendrecv_hint, 111
shishi_tgs_set_realm, 111
shishi_tgs_set_realmserver, 112
shishi_tgs_set_server, 111
shishi_tgs_tgtkt, 107
shishi_tgs_tgtkt_set, 107
shishi_tgs_tkt, 110
shishi_tgs_tkt_set, 111
shishi_tgsrep, 131
shishi_tgsreq, 113
shishi_tgsreq_rst, 113
shishi_ticket, 62
shishi_ticket_add_enc_part, 65
shishi_ticket_decrypt, 65
shishi_ticket_get_enc_part_etype, 64
shishi_ticket_parse, 90
shishi_ticket_print, 89
shishi_ticket_read, 90
shishi_ticket_realm_get, 63
shishi_ticket_realm_set, 64
shishi_ticket_save, 89
shishi_ticket_server, 62
shishi_ticket_set_enc_part, 64
shishi_ticket_set_server, 63
shishi_ticket_sname_set, 62
shishi_ticket_srealmserver_set, 63
Shishi_ticketflags, 42
shishi_time, 153
Shishi_tkt, 50
shishi_tkt, 67
shishi_tkt2, 67
shishi_tkt_authctime, 71

shishi_tkt_authtime, 71
shishi_tkt_build, 70
shishi_tkt_client, 68
shishi_tkt_client_p, 68
shishi_tkt_clientrealm, 69
shishi_tkt_clientrealm_p, 69
shishi_tkt_clientrealm_set, 69
shishi_tkt_decrypt, 74
shishi_tkt_done, 74
shishi_tkt_enckdcreppart, 66
shishi_tkt_enckdcreppart_set, 66
shishi_tkt_encticketpart, 66
shishi_tkt_encticketpart_set, 66
shishi_tkt_endctime, 72
shishi_tkt_endtime, 72
shishi_tkt_expired_p, 74
shishi_tkt_flags, 75
shishi_tkt_flags_add, 75
shishi_tkt_flags_set, 75
shishi_tkt_forwardable_p, 75
shishi_tkt_forwarded_p, 76
shishi_tkt_hw_authent_p, 78
shishi_tkt_initial_p, 77
shishi_tkt_invalid_p, 77
shishi_tkt_kdcrep, 66
shishi_tkt_key, 67
shishi_tkt_key_set, 67
shishi_tkt_keytype, 72
shishi_tkt_keytype_fast, 73
shishi_tkt_keytype_p, 73
shishi_tkt_lastreq, 70
shishi_tkt_lastreq_pretty_print, 70
shishi_tkt_lastreqc, 70
shishi_tkt_match_p, 83
shishi_tkt_may_postdate_p, 76
shishi_tkt_ok_as_delegate_p, 79
shishi_tkt_postdated_p, 77
shishi_tkt_pre_authent_p, 78
shishi_tkt_pretty_print, 68
shishi_tkt_proxiabile_p, 76
shishi_tkt_proxy_p, 76
shishi_tkt_realm, 68
shishi_tkt_renew_till, 72
shishi_tkt_renew_tillc, 72
shishi_tkt_renewable_p, 77
shishi_tkt_server, 73
shishi_tkt_server_p, 73
shishi_tkt_serverrealm_set, 69
shishi_tkt_startctime, 71
shishi_tkt_starttime, 71
shishi_tkt_ticket, 65
shishi_tkt_ticket_set, 65
shishi_tkt_transited_policy_checked_p, 78
shishi_tkt_valid_at_time_p, 74
shishi_tkt_valid_now_p, 74
Shishi_tkts, 50
shishi_tkts, 80
shishi_tkts_add, 80
shishi_tkts_add_ccache_file, 87
shishi_tkts_add_ccache_mem, 87
shishi_tkts_default, 79
shishi_tkts_default_ccache, 86
shishi_tkts_default_ccache_guess, 86
shishi_tkts_default_ccache_set, 86
shishi_tkts_default_file, 79
shishi_tkts_default_file_guess, 79
shishi_tkts_default_file_set, 79
shishi_tkts_default_to_file, 80
shishi_tkts_done, 83
shishi_tkts_expire, 81
shishi_tkts_find, 83
shishi_tkts_find_for_clientserver, 84
shishi_tkts_find_for_server, 84
shishi_tkts_from_ccache_file, 88
shishi_tkts_from_ccache_mem, 87
shishi_tkts_from_file, 83
shishi_tkts_get, 84
shishi_tkts_get_for_clientserver, 85
shishi_tkts_get_for_localservicepasswd, 86
shishi_tkts_get_for_server, 85
shishi_tkts_get_tgs, 85
shishi_tkts_get_tgt, 85
Shishi_tkts_hint, 49
Shishi_tkts_hintflags, 49
shishi_tkts_new, 81
shishi_tkts_nth, 80
shishi_tkts_print, 82
shishi_tkts_print_for_service, 81
shishi_tkts_read, 82
shishi_tkts_remove, 81
shishi_tkts_size, 80
shishi_tkts_to_file, 82
shishi_tkts_write, 82
Shishi_tr_type, 42
shishi_verbose, 57
shishi_verify, 170
SHISHI_VERSION, 40
shishi_warn, 57
shishi_x509ca_default_file, 250
shishi_x509ca_default_file_guess, 249
shishi_x509ca_default_file_set, 249
shishi_x509cert_default_file, 250
shishi_x509cert_default_file_guess, 250
shishi_x509cert_default_file_set, 250
shishi_x509key_default_file, 251
shishi_x509key_default_file_guess, 250
shishi_x509key_default_file_set, 251
shishi_xalloc_die, 251
