NAME

krb5_c_block_size, **krb5_c_decrypt**, **krb5_c_encrypt**, **krb5_c_encrypt_length**, **krb5_c_enctype_compare**, **krb5_c_get_checksum**, **krb5_c_is_coll_proof_cksum**, **krb5_c_is_keyed_cksum**, **krb5_c_keylength**, **krb5_c_make_checksum**, **krb5_c_make_random_key**, **krb5_c_set_checksum**, **krb5_c_valid_cksumtype**, **krb5_c_valid_enctype**, **krb5_c_verify_checksum**, **krb5_c_checksum_length** **-** Kerberos 5 crypto API

LIBRARY

Kerberos 5 Library (libkrb5, -lkrb5)

SYNOPSIS

#include <krb5.h>

krb5_error_code
krb5_c_block_size(*krb5_context context*, *krb5_enctype enctype*, *size_t *blocksize*);

krb5_error_code
krb5_c_decrypt(*krb5_context context*, *const krb5_keyblock key*, *krb5_keyusage usage*,
*const krb5_data *ivec*, *krb5_enc_data *input*, *krb5_data *output*);

krb5_error_code
krb5_c_encrypt(*krb5_context context*, *const krb5_keyblock *key*, *krb5_keyusage usage*,
*const krb5_data *ivec*, *const krb5_data *input*, *krb5_enc_data *output*);

krb5_error_code
krb5_c_encrypt_length(*krb5_context context*, *krb5_enctype enctype*, *size_t inputlen*, *size_t *length*);

krb5_error_code
krb5_c_enctype_compare(*krb5_context context*, *krb5_enctype e1*, *krb5_enctype e2*,
*krb5_boolean *similar*);

krb5_error_code
krb5_c_make_random_key(*krb5_context context*, *krb5_enctype enctype*, *krb5_keyblock *random_key*);

krb5_error_code
krb5_c_make_checksum(*krb5_context context*, *krb5_cksumtype cksumtype*, *const krb5_keyblock *key*,
krb5_keyusage usage, *const krb5_data *input*, *krb5_checksum *cksum*);

krb5_error_code
krb5_c_verify_checksum(*krb5_context context*, *const krb5_keyblock *key*, *krb5_keyusage usage*,
*const krb5_data *data*, *const krb5_checksum *cksum*, *krb5_boolean *valid*);

krb5_error_code
krb5_c_checksum_length(*krb5_context context*, *krb5_cksumtype cksumtype*, *size_t *length*);

krb5_error_code
krb5_c_get_checksum(*krb5_context context*, *const krb5_checksum *cksum*, *krb5_cksumtype *type*,
*krb5_data **data*);

krb5_error_code
krb5_c_set_checksum(*krb5_context context*, *krb5_checksum *cksum*, *krb5_cksumtype type*,
*const krb5_data *data*);

krb5_boolean
krb5_c_valid_enctype(*krb5_enctype*, *etype"*);

krb5_boolean
krb5_c_valid_cksumtype(*krb5_cksumtype ctype*);

krb5_boolean
krb5_c_is_coll_proof_cksum(*krb5_cksumtype ctype*);

krb5_boolean
krb5_c_is_keyed_cksum(*krb5_cksumtype ctype*);

krb5_error_code
krb5_c_keylengths(*krb5_context context*, *krb5_enctype enctype*, *size_t *inlength*, *size_t *keylength*);

DESCRIPTION

The functions starting with krb5_c are compat functions with MIT kerberos.

The krb5_enc_data structure holds and encrypted data. There are two public accessable members of krb5_enc_data. enctype that holds the encryption type of the data encrypted and ciphertext that is a *krb5_data* that might contain the encrypted data.

krb5_c_block_size() returns the blocksize of the encryption type.

krb5_c_decrypt() decrypts *input* and store the data in *output.* If *ivec* is NULL the default initialization vector for that encryption type will be used.

krb5_c_encrypt() encrypts the plaintext in *input* and store the ciphertext in *output*.

krb5_c_encrypt_length() returns the length the encrypted data given the plaintext length.

krb5_c_enctype_compare() compares to encryption types and returns if they use compatible encryption key types.

krb5_c_make_checksum() creates a checksum *cksum* with the checksum type *cksumtype* of the data in *data*. *key* and *usage* are used if the checksum is a keyed checksum type. Returns 0 or an error code.

krb5_c_verify_checksum() verifies the checksum of *data* in *cksum* that was created with *key* using the key usage *usage*. *verify* is set to non-zero if the checksum verifies correctly and zero if not. Returns 0 or an error code.

krb5_c_checksum_length() returns the length of the checksum.

krb5_c_set_checksum() sets the krb5_checksum structure given *type* and *data*. The content of *cksum* should be freeed with **krb5_c_free_checksum_contents**().

krb5_c_get_checksum() retrieves the components of the krb5_checksum. structure. *data* should be free with **krb5_free_data**(). If some either of *data* or *checksum* is not needed for the application, NULL can be passed in.

krb5_c_valid_enctype() returns true if *etype* is a valid encryption type.

krb5_c_valid_cksumtype() returns true if *ctype* is a valid checksum type.

krb5_c_is_keyed_cksum() return true if *ctype* is a keyed checksum type.

krb5_c_is_coll_proof_cksum() returns true if *ctype* is a collision proof checksum type.

krb5_c_keylengths() return the minimum length (*inlength*) bytes needed to create a key and the length (*keylength*) of the resulting key for the *enctype*.

SEE ALSO

krb5(3), krb5_create_checksum(3), krb5_free_data(3), kerberos(8)